Stony Brook University

Combinatorics and Complexity in Geometric Visibility Problems

A Dissertation Presented
by
Justin G. Iwerks
to
The Graduate School
in Partial Fulfillment of the
Requirements
for the Degree of

Doctor of Philosophy
in
Applied Mathematics and Statistics
(Operations Research)

Stony Brook University
August 2012

Stony Brook University

The Graduate School

Justin G. Iwerks

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation.

Joseph S. B. Mitchell - Dissertation Advisor
Professor, Department of Applied Mathematics and Statistics

Esther M. Arkin - Chairperson of Defense
Professor, Department of Applied Mathematics and Statistics

Steven Skiena
Distinguished Teaching Professor, Department of Computer Science

Jie Gao - Outside Member
Associate Professor, Department of Computer Science

Charles Taber
Interim Dean of the Graduate School

Abstract of the Dissertation

Combinatorics and Complexity in Geometric Visibility Problems

by
Justin G. Iwerks

Doctor of Philosophy
in
Applied Mathematics and Statistics
(Operations Research)

Stony Brook University
2012

Geometric visibility is fundamental to computational geometry and its applications in areas such as robotics, sensor networks, CAD, and motion planning. We explore combinatorial and computational complexity problems arising in a collection of settings that depend on various notions of visibility.

We first consider a generalized version of the classical art gallery problem in which the input specifies the number of reflex vertices r and convex vertices c of the simple polygon ($n = r + c$). This additional information better characterizes the shape of the polygon. Through a lower bound construction, tight combinatorial bounds for coverage are achieved for all $r \geq 0$ and $c \geq 3$.

The combinatorics of guarding polyominoes and other polyforms are studied in terms of m, the number of cells, as opposed to the traditional parameter n. Various visibility models and guard types are considered. We establish that finding a minimum cardinality guard set for covering a polyomino is NP-hard.

We introduce an algorithm for constructing a spiral serpentine polygonization of a set of $n \geq 3$ points in the plane. The algorithm's behavior can be viewed as incrementally appending a visible triangle to the triangulation constructed so far.

We consider beacon-based point-to-point routing and coverage problems. A beacon b is a point that can be activated to effect a gravitational pull toward itself in a polygonal domain. Algorithms are given for computing the attraction region of b and finding a minimum size set of beacons to route from a source s to a destination t given a finite set of candidate beacon locations.

We show that finding a minimum cardinality set of beacons to cover a simple polygon or conduct certain types of routing in a simple polygon is NP-hard.

Contents

List of Figures

Acknowledgments

I am indebted to my advisor, Joseph S. B. Mitchell, for helping me develop into an independent mathematical researcher. I must also thank Joe for introducing me to the wonderful world of computational geometry. I hope to inspire my own students with the rich intensity of excitement and passion he has for this field.

I would like to thank James Glimm and Xiaolin Li for their valuable guidance and support while I began my first research projects.

I am grateful for my colleagues and co-authors including Esther Arkin, Therese Biedl, Michael Biro, Brian Fix, Jie Gao, Mohammed T. Irfan, Tulin Kaman, Ryan Kaufman, Joondong Kim, Irina Kostitsyna, Hyunkyung Lim, Alan Tucker, Shang Yang and Yan Yu.

I would like to thank Amy Picard Winston, my superb high school physics teacher, for inspiring me to pursue physics and mathematics in college.

I am thankful for my parents and siblings who always made learning fun growing up and encouraged me to pursue my academic interests.

I would like to thank my wife, Rebecca, for her steadfast patience and support. Despite moving three times, years of long work commutes, and budgeting with a graduate student's income, she has gracefully stuck by my side. Thanks also to our daughter, Cambria, for providing uncountable amounts of joy.

Completing my doctoral degree is nothing short of a grand and improbable victory that I could never have accomplished alone. I therefore wish to give thanks to Jesus Christ and close with the following quote:

Then Samuel took a stone and set it up between Mizpah and Shen. He named it Ebenezer, saying, "Thus far has the LORD helped us."

- 1 Samuel 7:12

List of Publications

[1] T. Biedl, M. T. Irfan, J. Iwerks, J. Kim, and J. S. B. Mitchell. Guarding polyominoes. In *Proc. of the 27th Annual Symposium on Computational Geometry (SoCG 2011)*, pages 387–396, Paris, France, 2011. Association for Computing Machinery.

[2] T. Biedl, M. T. Irfan, J. Iwerks, J. Kim, and J. S. B. Mitchell. The art gallery theorem for polyominoes. *Discrete and Computational Geometry (to appear)*, 2012.

[3] M. Biro, J. Gao, J. Iwerks, I. Kostitsyna, and J. S. B. Mitchell. Beacon based routing and coverage. In *Proc. of the 21st Fall Workshop on Computational Geometry*, pages 16–17, New York, NY, November 2011.

[4] M. Biro, J. Gao, J. Iwerks, I. Kostitsyna, and J. S. B. Mitchell. Beacon based structures in polygonal domains. In *Abstracts of the 1st Computational Geometry: Young Researchers Forum*, Chapel Hill, NC, June 2012.

[5] M. T. Irfan, J. Iwerks, J. Kim, and J. S. B. Mitchell. Guarding polyominoes. In *Proc. of the 19th Fall Workshop on Computational Geometry*, pages 39–40, Medford, MA, November 2009.

[6] J. Iwerks. Guarding polyforms. In *Abstracts of the 1st Computational Geometry: Young Researchers Forum*, Chapel Hill, NC, June 2012.

[7] J. Iwerks and J. S. B. Mitchell. Spiral serpentine polygonization of a planar point set. In *Proc. of XIV Spanish Meeting on Computational Geometry*, pages 181–184, Alcalá de Henares, Spain, 2011. Centre de Recerca Matemática.

[8] J. Iwerks and J. S. B. Mitchell. The art gallery theorem for simple polygons in terms of the number of reflex and convex vertices. *Information Processing Letters (to appear)*, 2012.

[9] J. Iwerks and J. S. B. Mitchell. Spiral serpentine polygonization of a planar point set. *Lecture Notes in Computer Science (to appear)*, 2012.

[10] H. Lim, J. Iwerks, J. Glimm, and D. H. Sharp. Nonideal Rayleigh-Taylor mixing. *Proc. of the National Academy of Sciences of the United States of America*, 107(29):12786–12792, 2010.

[11] H. Lim, J. Iwerks, Y. Yu, J. Glimm, and D. H. Sharp. Verification and validation of a method for the simulation of turbulent mixing. *Physica Scripta*, T142:014014, 2010.

Chapter 1

Introduction

Geometric visibility plays a fundamental role in a host of problems in the fields of discrete and computational geometry. Applications ranging from motion planning to sensor networks and computer-aided design all depend critically on various models of visibility. We define the most common notion of visibility in a domain D, say a subset of $\mathbb{R}^n$, as follows: A point $q \in D$ is *visible* to a point $p \in D$ if $\overline{pq} \subset D$. We say that p 'sees' q and vice-versa (since the definition is symmetric). However, there are myriad generalizations of this definition that find appropriate application in a variety of settings. For instance, we may be interested in more robust notions of visibility in which q is visible to p only if p also sees a sufficiently large region containing q, or if the entire rectangle spanned by p and q is contained in D. Alternatively, perhaps we have a robot that can move in straight lines and make turns, but turning requires a great deal of energy. In this case, the set of points that may be visible or 'reachable' for the robot are the set of points it can arrive at using at most k turns or having a *link-distance* of k from the robot's starting location.

This thesis focuses on the combinatorics and computational complexity of a variety problems that intrinsically rely on various models of geometric visibility. The following two vignettes provide some motivational context for these visibility problems:

(a) A museum installs a security camera with a 360° viewing angle inside its modern art gallery. Which points in the gallery does the camera see? How many of these cameras should the museum install to ensure that every point (and hence every work of art!) is seen? Where should the

cameras be placed?

(b) A mobile robot located in the downtown area of a city receives a 'return home' message from a signal tower. The robot subsequently attempts to make a straight line path back to the tower. The route may involve moving along the boundaries of the domain as long as the robot's distance to the tower always decreases monotonically. Which locations in the city should the robot limit itself to so that it can be guaranteed to return to the tower without getting stuck elsewhere? In other words, which points are in the *attraction region* of the tower? Furthermore, if we were interested in routing a robot from a starting location s to a destination location t using signal towers as 'stepping stones' (robot i starts at s and visits towers b_1, b_2, ..., b_k before reaching its final destination t), where should they be placed?

In the first vignette, the notion of visibility is precisely the one described above: let p be the security camera and then find the locus of points Q that can be seen by p. Finding sometimes necessary and always sufficient bounds on the number of cameras needed to see or *cover* all of a domain modeled as a polygon with n vertices is often referred to as the *Art Gallery Problem*. It was first shown by Vasek Chvátal that $\lfloor \frac{n}{3} \rfloor$ point guards are sometimes necessary and always sufficient to cover a simple polygon with n vertices. Art gallery problems have since developed into a significant area of research in computational geometry and theoretical computer science with numerous articles, several surveys and an entire book dedicated to the problem and its generalizations [42, 47, 49].

In Chapter 2, we study one such generalization in which the input specifies the number of reflex vertices r and convex vertices c of the simple polygon, where $n = r + c$. Specification of this additional information is motivated by attempting to provide a better description of the 'shape' of the polygon. For instance, a polygon with 99 vertices may require 33 point guards according to Chvátal's art gallery theorem, but if only 3 of these vertices are convex, then we actually have a pseudotriangle that never requires more than 2 point guards! Tight combinatorial bounds for this problem have previously been shown when $0 \leq r \leq \lfloor \frac{c}{2} \rfloor$ and $r \geq 5c - 12$. We give a lower bound construction that matches the $\lfloor \frac{n}{3} \rfloor$ sufficiency condition from the art gallery theorem when $\lfloor \frac{c}{2} \rfloor < r < 5c - 12$, thus providing tight combinatorial bounds for all $r \geq 0$ and $c \geq 3$.

Chapters 3 and 4 examine a significantly different kind of art gallery problem in which our domain consists of a connected union of m edge- or face-aligned copies of some base shape, such as a square, equilateral triangle, or cube. In particular, we study the combinatorics of guarding polyominoes (connected unions of edge-aligned unit squares) in terms of the parameter m, in contrast with the traditional parameter n, the number of vertices of P. We show that $\lfloor \frac{m+1}{3} \rfloor$ *point guards* are always sufficient and sometimes necessary to cover an m-polyomino. When pixels act like guards (*pixel guards*), we prove that $\left\lfloor \frac{3m}{11} \right\rfloor + 1$ guards are sufficient and sometimes necessary to cover an m-polyomino. For the 3D analog of a polyomino, a *polycube*, we show that $\lfloor \frac{m+1}{3} \rfloor$ point guards are sometimes necessary and always sufficient, matching the tight bounds previously shown for guarding an m-polyomino. Other combinatorial bounds are given for guarding polycubes with *voxel guards* (guards that occupy entire voxels of the polycube) as well as connected unions of equilateral triangles (*polyiamonds*) and regular hexagons (*polyhexes*) using both point and pixel guards. We establish that determining the minimum number of guards required to cover a given m-polyomino is NP-hard. We also provide polynomial-time algorithms to solve exactly some special cases in which a polyomino is 'thin' (no cycles of length 4 or more in the dual graph).

In Chapter 5 we devise an algorithm for constructing a *spiral serpentine polygonization* of a set S of $n \geq 3$ points in the plane. A polygonization of a planar point set S is a simple polygon having S as the set of its vertices. This polygonization is spiral because it has exactly one polygonal chain of convex vertices and at most one polygonal chain of reflex vertices. The serpentine property implies that the polygonization has a triangulation whose dual graph is a path. Our algorithm simultaneously gives a triangulation of the constructed polygon at no extra cost, runs in $O(n \log n)$ time and uses $O(n)$ space. Both the runtime and space performance are shown to be optimal. The algorithm's behavior can be viewed as incrementally appending a visible triangle to the triangulation constructed so far.

In the second vignette, we have a much different model of visibility in which a robot is considered visible or *attracted* to a signal tower if the robot can make a straight line path to the tower, perhaps involving intermediate sliding along the boundary of the domain, without getting stuck somewhere along the way. This model applied to coverage and routing problems in polygonal domains is studied in Chapter 6. These problems are motivated by sensor network applications and we generally refer to the signal towers as *beacons*. We show that $\left\lfloor \frac{n}{2} \right\rfloor - 1$ beacons are sometimes necessary and always sufficient to route

between any pair of points in P. We demonstrate that finding a minimum cardinality set of beacons to route from any source point $s \in P$ to a given destination $t \in P$ or from a particular source to any given destination is NP-hard. We also show how to efficiently compute the attraction region of a beacon and use this to establish a polynomial-time algorithm for routing from a point s to a point t using a discrete set of candidate beacons. We close by showing that it is NP-hard to find a minimum cardinality set of beacons to cover a simple polygon.

Each chapter provides a more thorough introduction along with a review of related work found in the literature. We close each chapter with a conclusion featuring open problems for further investigation.

Chapter 2

The Art Gallery Theorem for Simple Polygons in Terms of the Number of Reflex and Convex Vertices[1]

2.1 Introduction

The problem of determining the minimum number of point guards sufficient to cover the interior of an art gallery represented by a simple polygon with n vertices was originally posed by Victor Klee in 1973. The solution to this problem was first given by Vasek Chvátal [17], who proved that $\lfloor \frac{n}{3} \rfloor$ guards are sometimes necessary and always sufficient to cover an n-vertex polygon. Art gallery problems have become a significant area of study in computational geometry, as they are not only of theoretical interest, but also play a central role in visibility problems arising in robotics, digital model capture, sensor networks, motion planning, vision, and computer-aided design [47, 49].

The combinatorial complexity of a simple polygon is naturally specified in terms of the number, n, of vertices. Vertices can be distinguished into two categories - *reflex* vertices (having internal angle greater than π), and *convex* vertices (having internal angle at most π). For a given simple polygon P, let $g(P)$ denote the minimum number of guards required to cover P. We define the *guard number* $G(r, c)$ to be the maximum value of $g(P)$, over all simple polygons P having exactly r reflex vertices and c convex vertices.

[1] This chapter is based on work joint with Joseph S. B. Mitchell [32].

In this note, we establish a refined art gallery theorem, giving exactly the function $G(r, c)$ for the guard number in terms of the number, $r \geq 0$, of reflex vertices and the number, $c \geq 3$, of convex vertices of P ($n = r + c$).

2.2 Previous Work

O'Rourke [42] proved that r guards are sometimes necessary and always sufficient to cover a simple polygon P having $r \geq 1$ reflex vertices. The upper bound readily comes from Chazelle's convex partitioning [14], while the lower bound is demonstrated through O'Rourke's *shutter* polygons (Figure 2.1). O'Rourke considered the art gallery problem solely in terms of the variable r, without consideration of the number, c, of convex vertices required to realize shutter polygons. In fact, shutter polygons require approximately twice as many convex vertices as reflex vertices. In our notation, O'Rourke showed that

$$G(r, c) = r, \quad \text{if } 1 \leq r \leq \left\lfloor \frac{c}{2} \right\rfloor.$$

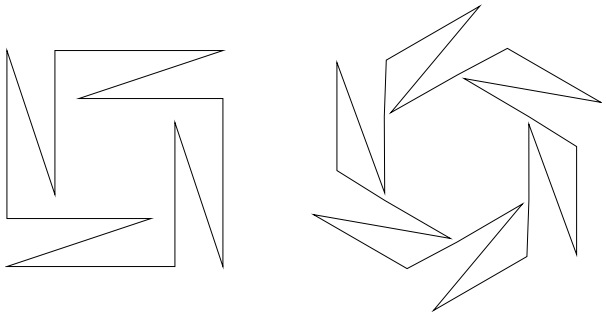


Figure 2.1: Shutter polygons require r guards

More recently, Addario-Berry et al. [2] proved that $2c - 4$ guards are sometimes necessary and always sufficient to cover a simple polygon P with c convex vertices. However, this tight bound is solely a function of the variable c. The sometimes necessary condition given in their paper is a *pseudotriangle chain* (Figure 2.2) that naturally requires a certain amount of reflex vertices in order to be realized. Inspection of these pseudotriangle chains reveals that $r \geq 5c - 12$ reflex vertices are required to realize it. Combined with the result

of O'Rourke and the trivial observation that $G(r, c) = 1$ when $r = 0$, we have

$$G(r,c) = \begin{cases} 1, & \text{if } r = 0 \\ \text{r}, & \text{if } r \leq \lfloor \frac{c}{2} \rfloor \\ \text{2c - 4}, & \text{if } r \geq 5c - 12 \end{cases}$$

The question remains as to what $G(r,\ c)$ is when $\lfloor \frac{c}{2} \rfloor < r < 5c - 12$. In Section 2.3 we answer this question by constructing polygons that interpolate between Addario-Berry et al.'s pseudotriangle chains and O'Rourke's shutter polygons.

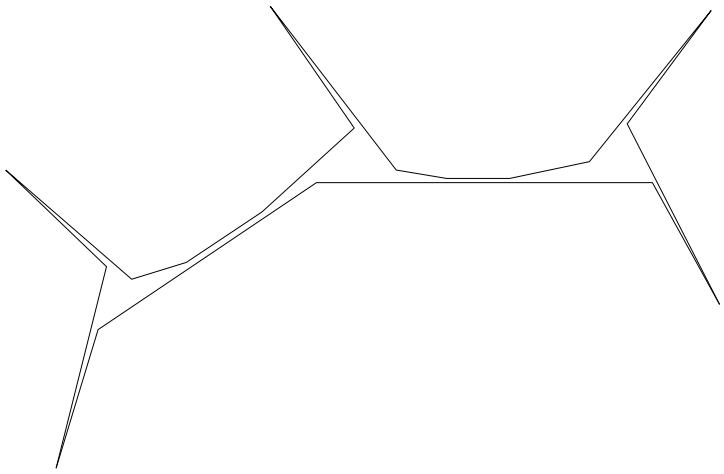


Figure 2.2: Pseudotriangle chain polygons require $2c - 4$ guards. In the example shown here, $c = 5$, $r = 13$ and 6 guards are required: 2 for each pseudotriangle.

2.3 Lower Bound Constructions

In this section we complete the full specification of the guard number function G by proving the following theorem:

Theorem 2.3.1. *Given a simple polygon P with r reflex vertices and c convex vertices ($n = r + c$), $2c - 4 - \lceil \frac{5c-12-r}{3} \rceil = \lfloor \frac{n}{3} \rfloor$ point guards are sometimes necessary and always sufficient to cover P when $\lfloor \frac{c}{2} \rfloor < r < 5c - 12$.*

Proof. To establish this result, we present an algorithm that generates a lower bound example for each r and c that satisfy $\lfloor \frac{c}{2} \rfloor < r < 5c - 12$. Since, by the standard art gallery theorem, $\lfloor \frac{n}{3} \rfloor$ guards are always sufficient, our bound is tight. The algorithm involves two *sweeps* of reflex vertex removals: the first begins with a pseudotriangle chain and terminates with what we will call a *spiny back* polygon, and the second sweep transforms our spiny back polygon into a shutter polygon (see Figure 2.3). We observe that spiny back polygons require a guard for each spine.

We can think of the original pseudotriangle chain as the union of two

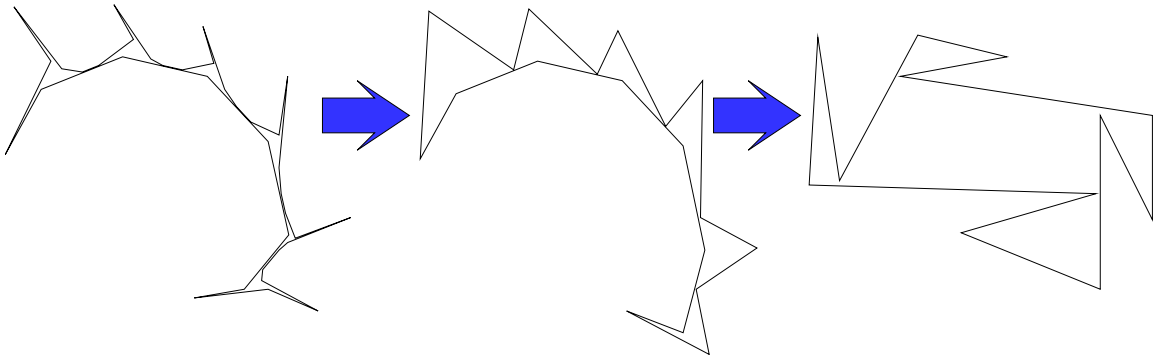


Figure 2.3: Two sweeps are carried out in our lower bound constructions by removing reflex vertices in a particular order. The first sweep transforms a pseudotriangle chain into a spiny back polygon and the second transforms the spiny back polygon into a shutter polygon.

polygonal chains. We let T denote the top chain and B denote the bottom chain, whose vertices are all reflex, except its convex vertex endpoints, which are shared by T and B. In order to avoid notation overuse, in our description we let γ denote the number of convex vertices of P, since "c_i" will be used to denote the specific convex vertices. We use the following label conventions, going from left to right along each chain (Figure 2.4):

$$\begin{aligned} T &= (c_1, r_1, c_2, r_{2,1}, r_{2,2}, r_{2,3}, r_{2,4}, c_3, r_{3,1}, r_{3,2}, r_{3,3}, r_{3,4}, ..., c_{\gamma-1}, r_{\gamma-1}, c_\gamma) \\ B &= (c_1, r_{B,1}, r_{B,2}, ..., r_{B,\gamma-2}, c_\gamma) \end{aligned}$$

The first sweep involves the removal of reflex vertices of T in the following

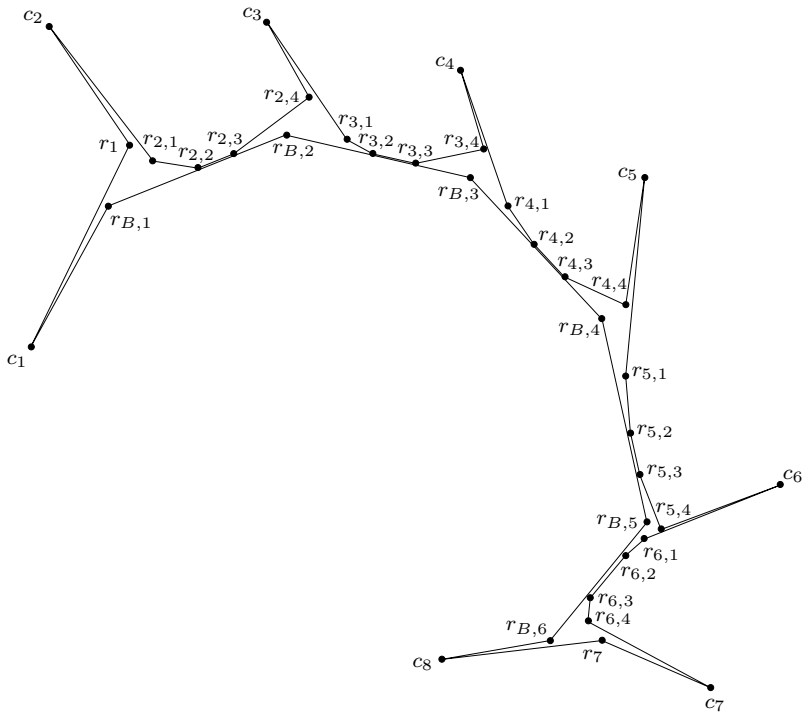


Figure 2.4: Labeling of a pseudotriangle chain with 8 convex vertices

order:

First Sweep:

$r_1,$

$r_{2,1}, r_{2,2}, r_{2,4},$

$r_{3,1}, r_{3,2}, r_{3,4},$

$\vdots$

$r_{\gamma-2,1}, r_{\gamma-2,2}, r_{\gamma-2,4},$

$r_{\gamma-1}$

It can be observed that removing the vertices listed in each row (except the last) of the above first sweep causes the number of guards required to cover the resulting polygon to decrease by 1. This behavior matches the functional values given by $\lfloor \frac{n}{3} \rfloor$.

After the first sweep is complete, we obtain a spiny back polygon. We then proceed with the second sweep, which removes certain reflex vertices from both

T and B and also slides some vertices to form barbs of the shutter polygon:

Second Sweep:

$r_{B,1}, r_{3,3}$, slide $r_{4,3}$ near the interior of $\overline{r_{2,3}c_3}$,

$r_{B,2}, r_{B,3}, r_{5,3}$, slide $r_{6,3}$ near the interior of $\overline{r_{4,3}c_5}$,

$r_{B,4}, r_{B,5}, r_{7,3}$, slide $r_{8,3}$ near the interior of $\overline{r_{6,3}c_7}$,

$\vdots$

$r_{B,j}, r_{B,j+1}, r_{j+3,3}$, slide $r_{j+4,3}$ near the interior of $\overline{r_{j+2,3}c_{j+3}}$ (j even),

$\vdots$

if γ is odd

$r_{B,\gamma-5}, r_{B,\gamma-4}, r_{\gamma-2,3}$

else

$r_{B,\gamma-4}, r_{B,\gamma-3}$

Again, the number of guards required to cover the resulting polygon we construct upon the removal of the vertices contained in each row (except the last) of the second sweep decreases by 1. We can think of the last row of the first sweep and the first row of the second sweep as one row with 3 reflex vertices being removed. Since, through both sweeps, we maintain the ratio of 3:1 between the number of vertices removed to the reduction in the number of guards required, we continue to match the functional values of $\lfloor \frac{n}{3} \rfloor$ and thus provide tight combinatorial bounds for all r and c in the range $\lfloor \frac{c}{2} \rfloor < r < 5c - 12$. $\square$

We now have tight combinatorial bounds for all $r \geq 0$ and $c \geq 3$:

Theorem 2.3.2. *Let P be a simple polygon with r reflex vertices and c convex vertices ($n = r + c$). Then,*

$$G(r,c) = \begin{cases} 1, & \text{if } r = 0 \\ r, & \text{if } r \leq \lfloor \frac{c}{2} \rfloor \\ \lfloor \frac{n}{3} \rfloor, & \text{if } \lfloor \frac{c}{2} \rfloor < r < 5c - 12 \\ 2c-4, & \text{if } r \geq 5c - 12. \end{cases}$$

In order to illustrate the behavior of the two sweeps used to construct the lower bound examples, we will consider the case in which $c = 7$ and $5c - 12 = 23 > r > 3 = \lfloor \frac{c}{2} \rfloor$ with the aid of Figure 2.5. Note that, for clarity, the vertex labels have been omitted from Figure 2.5. Beginning with a pseudotriangle

chain on 7 convex vertices, 23 reflex vertices, and requiring 10 guards as shown in Figure 2.5a., we first remove r_1, yielding the polygon depicted in Figure 2.5b. The removal of this reflex vertex eliminates the need for one of the guards in the leftmost pseudotriangle. The remaining pseudotriangles still each require two guards, for a total of 9. Since $n = r + c = 22 + 7 = 29$ and $\lfloor \frac{29}{3} \rfloor = 9$, this guard set size is tight.

We then begin removing vertices along the reflex chain between c_2 and c_3. First, we remove $r_{2,1}$, which yields the polygon depicted in Figure 2.5c. The removal of this vertex creates our first spine, which requires one guard. The remaining pseudotriangles still require 2 guards each, so we maintain a guard set of size 9. This matches $\lfloor \frac{28}{3} \rfloor = 9$.

Secondly, we remove $r_{2,2}$, giving us the polygon shown in Figure 2.5d. Despite this reflex vertex's removal, we still require 9 guards, and, since $\lfloor \frac{27}{3} \rfloor = 9$, we have agreement.

The third reflex vertex we remove from this chain is $r_{2,4}$, which leaves us with the polygon shown in Figure 2.5e. This final vertex removal from the chain between c_2 and c_3 turns the second (original) pseudotriangle into a spine-like structure that requires just one guard. Hence, we need only 8 guards at this juncture, giving us agreement with $\lfloor \frac{26}{3} \rfloor = 8$. The behavior of the first sweep continues in this fashion.

After the first sweep is completed, we arrive at the spiny back polygon depicted in Figure 2.5f, having $n = r + c = 9 + 7 = 16$ vertices and requiring 5 guards as shown. Since $\lfloor \frac{16}{3} \rfloor = 5$, our guard set size is tight. The second sweep begins with the removal of $r_{B,1}$, yielding the polygon shown in Figure 2.5g, with 15 vertices and needing 5 guards, which matches $\lfloor \frac{15}{3} \rfloor = 5$.

The next vertex removed during the second sweep is $r_{3,3}$, followed by moving $r_{4,3}$ near the interior of $\overline{r_{2,3}c_3}$, as depicted in Figure 2.5h. This forms the second *shutter* of the shutter polygon we are working toward. Four guards are required to cover the resulting polygon, agreeing with the $\lfloor \frac{14}{3} \rfloor = 4$ upper bound.

Subsequently, $r_{B,2}$ and $r_{B,3}$ are removed from the bottom polygonal chain, as shown in Figures 2.5i and 2.5j, respectively. Four guards are required in each case, matching our upper bounds of $\lfloor \frac{13}{3} \rfloor = 4$ and $\lfloor \frac{12}{3} \rfloor = 4$.

We complete the second sweep by removing $r_{5,3}$, which gives us a polygon requiring 3 guards, as shown in Figure 2.5k. Since $\lfloor \frac{11}{3} \rfloor = 3$, we again have agreement with the upper bound.

Finally, we may observe that the removal of $r_{B,3}$ allows us to form a shutter polygon, depicted in Figure 2.5l, as desired.

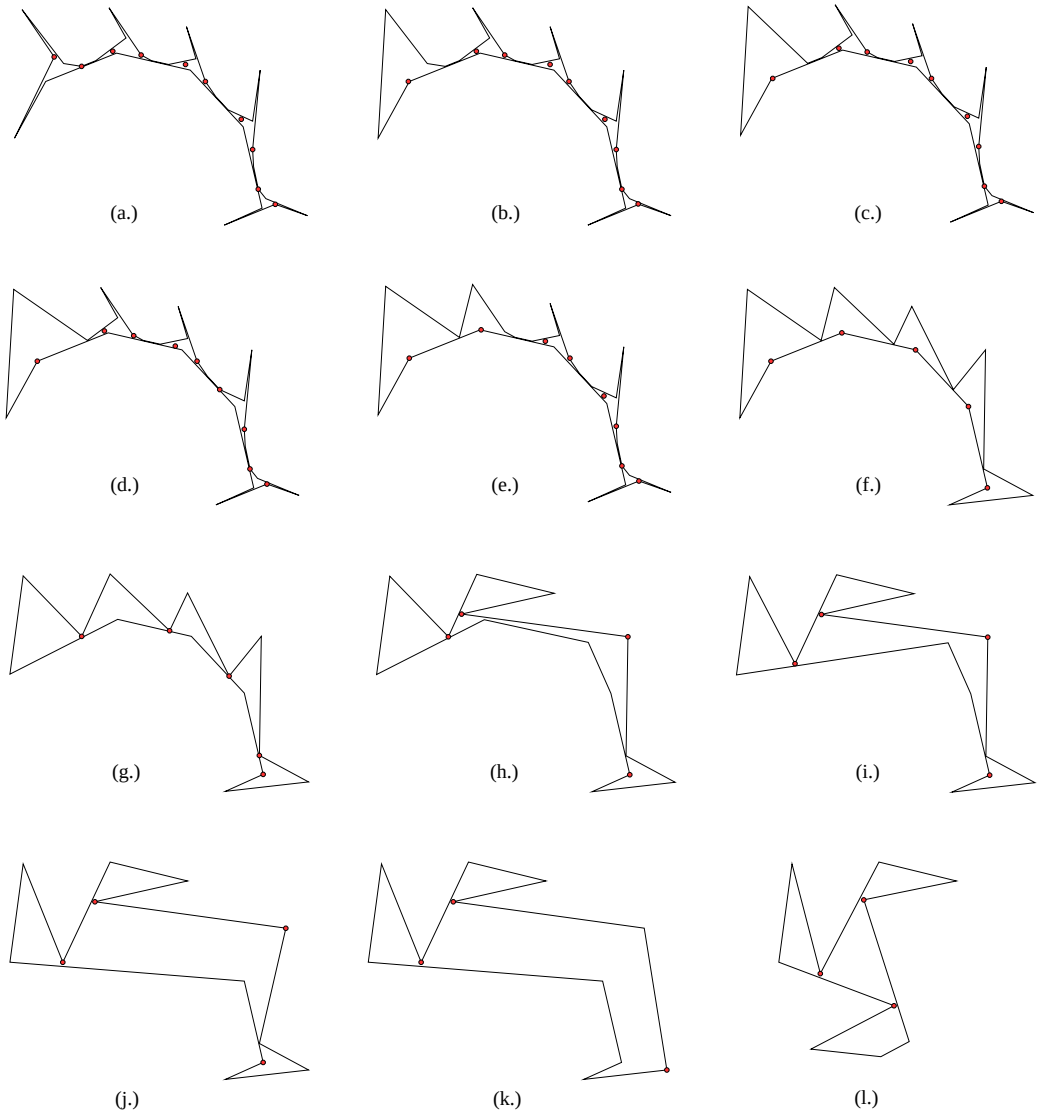


Figure 2.5: An illustration of our lower bound constructions using 7 convex vertices. The shaded points represent possible guard sets of minimum size in each figure.

2.4 Conclusion

Through the systematic construction of lower bound examples, we have demonstrated that $G(r, c) = \lfloor \frac{n}{3} \rfloor$ for $\lfloor \frac{c}{2} \rfloor < r < 5c - 12$. Combined with previous results of O'Rourke and Addario-Berry et al., this gives tight combinatorial bounds on the number of guards required to guard any simple polygon in terms of r and c, for all cases.

An interesting extension of this result would be to give combinatorial bounds for polygons with holes. In other words, if P has r reflex vertices, c convex vertices and h holes, what is the worst-case minimum number of guards to cover P, as a function of r, c, and h? A related question is to determine the minimum number of guards to cover P, as a function of r, c, and the number of reflex chains of P.

Chapter 3

Guarding Polyominoes[1]

3.1 Introduction

Victor Klee (1973) posed the problem of determining the minimum number of guards sufficient to cover the interior of an art gallery modeled as a simple polygon P with n vertices. The solution, first given by Vasek Chvátal, is that $\lfloor \frac{n}{3} \rfloor$ guards are sometimes necessary and always sufficient to cover a polygon possessing n vertices [42]. This original problem has since grown into a significant area of study in computational geometry and computer science. Art gallery problems are of theoretical interest but also play a central role in visibility problems arising in applications in robotics, digital model capture, sensor networks, motion planning, vision, and computer-aided design [47, 49].

We explore the art gallery problem when the given gallery P is an m-*polyomino*, a polyform whose cells are integral unit squares. (In other words, an m-polyomino P is the union of m (closed) integral unit squares such that the interior of P is connected.) We refer to the unit squares as *pixels*. An example with $m = 29$ is shown in Figure 3.1. We often write (m) for an m-polyomino. For example, (7) refers to any 7-polyomino. The *dual graph* of an m-polyomino P has a node for each pixel of P and an edge joining two nodes that correspond to edge-adjacent pixels. A polyomino P is *simple* if it has no

[1]This chapter is based on work joint with Therese Biedl at University of Waterloo and Mohammed T. Irfan, Joondong Kim, and Joseph S. B. Mitchell at Stony Brook University. A preliminary version appeared in *Proceedings of the 27th Annual Symposium on Computational Geometry (SoCG 2011)* [31] and a journal article focusing on the combinatorial results with point guards appears in *Discrete and Computational Geometry* [33].

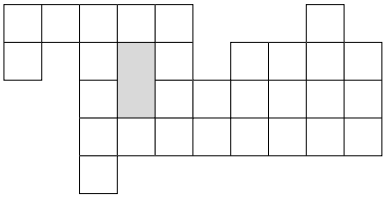


Figure 3.1: A polyomino with 29 pixels. The grey region is a hole.

holes: every minimal cycle in its dual graph is a 4-cycle (i.e., the complement of the interior of P is connected).

A point $a \in P$ *covers* (or *sees*) a point $b \in P$ if the line segment $ab \subset P$. (Since P is closed, possibly ab contains points on the boundary, ∂P.) A pixel p of P *covers* a point $b \in P$ if there is a point a inside of p that covers b.

We introduce two types of guards: *point guards* and *pixel guards*. As its name suggests, a point guard occupies a point location in P. A pixel guard occupies an entire pixel. Note that, a point/pixel guard can see an unlimited distance in every direction. If a point $b \in P$ is covered by a point (pixel) p, then p is a point (pixel) guard of b.

We let $g(P)$ denote the *guard number* of P: $g(P)$ is the minimum number of guards required to cover all of a given polyomino P. Then we define $G(m)$ to be the maximum value of $g(P)$ over all m-polyominoes P. Our main combinatorial question is that of determining upper and lower bounds on $G(m)$.

There are possible alternative models of visibility that one can study. One such model is *r-visibility*, where two points can see each other iff the axis-parallel rectangle defined by them is a subset of P. Another model, the *all-or-nothing* model, considers a pixel p to be guarded only if a single point guard a (or point a inside a pixel guard) sees all points of p (i.e., $ab \subset P$, for all $b \in P$). It is easy to see that a cover in the r-visibility model is a cover in the all-or-nothing model, which in turn is a cover in the unrestricted model, and none of these implications hold in reverse in general. See Figure 3.2 for illustrations of these models.

Previous work. While $\lfloor \frac{n}{3} \rfloor$ guards are sufficient and sometimes necessary to cover a simple polygon with n vertices, $\lfloor \frac{n}{4} \rfloor$ are sufficient and sometimes necessary to cover an orthogonal polygon [27, 29, 34, 41]. It is NP-hard to find the minimum number of guards for covering either a general simple polygon [37] or an orthogonal polygon [45]. Even covering the vertices of an orthogonal

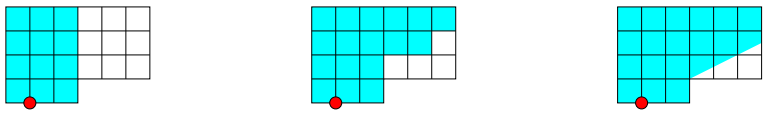


Figure 3.2: The three models of visibility. From left to right: r-visibility model, all-or-nothing model and unrestricted model.

polygon has been shown to be NP-hard [36].

While no approximation algorithm with better than the naive factor $\lfloor \frac{n}{3} \rfloor$ is known for placing the fewest guards at arbitrary points in a simple polygon, there are results for approximation of optimal guard placement in polygons by restricting the set of candidate guards. For instance, if we restrict guards to lie only on vertices of a polygon or grid points, logarithmic approximations are achievable based on set cover [19, 23, 24].

In orthogonal polygons, Nilsson [40] gives an algorithm to compute $O(OPT^2)$ guards, based on the constant-factor approximation for guarding 1.5D terrains [6]. Practical methods and heuristics for placing guards in general polygonal domains have been investigated by Amit et al. [4].

Of particular interest for guarding rectilinear polygons is a result of Worman and Keil [50], which applies in the r-visibility model: They show that a minimum guard cover in this model can be found in polynomial time. The algorithm runs in time $\tilde{O}(n^{17})$ and requires computing a maximum independent set in a perfect graph as a sub-routine.

Our results.

1. Combinatorial Bounds: For point guards, we show that $\lfloor \frac{m+1}{3} \rfloor$ guards are always sufficient and sometimes necessary to cover an m-polyomino (possibly with holes). For pixel guards, we show that $\left\lfloor \frac{3m}{11} \right\rfloor + 1$ guards are always sufficient and sometimes necessary to cover an m-polyomino.[2]

   The necessity bounds are in the unrestricted visibility model (and hence also hold in the all-or-nothing and the r-visibility model.) The sufficiency bounds are constructive and yield a guard set that is in the (most restrictive) r-visibility model. See Corollary 3.2.7 and Theorem 3.2.8.

[2]More precisely, our lower bound construction shows that $f(m) = \lfloor \frac{m+1}{11} \rfloor + \lfloor \frac{m+5}{11} \rfloor + \lfloor \frac{m+9}{11} \rfloor$ pixel guards are sometimes necessary. For certain values of m this can differ from $\left\lfloor \frac{3m}{11} \right\rfloor + 1$ by at most 1. Val Pinciu outlines how to obtain a matching upper bound of $f(m)$ for all m in [43].

2. Hardness: We show that determining an optimal guard placement in a simple m-polyomino is NP-hard, even in the all-or-nothing model. See Theorem 3.3.1. Recall that this problem is polynomial for point guards in the r-visibility model [50].

3. Algorithms for Special Cases: We consider the special case of a *thin* m-polyomino, for which the dual graph is a tree. In particular, we show that the algorithm by Worman and Keil (for optimal cover in the r-visibility model) in fact yields an optimal cover even in the all-or-nothing model, but not in the general model. We give simple optimal algorithms for guarding with point and pixel guards when the dual is a path.

3.2 Combinatorial Bounds

3.2.1 Necessary Conditions

Figure 3.3 shows a polyomino analog of Chvátal's comb, illustrating that, for $m \geq 2$, $\lfloor \frac{m+1}{3} \rfloor$ guards are sometimes necessary to cover an m-polyomino.

For the pixel guard case, we can create a polyomino consisting of $22k + 13$ pixels (for any $k \geq 0$) that requires $6k + 4$ pixel-guards. Figure 3.4 illustrates the case $k = 1$; polyominoes for larger k are obtained by attaching $k-1$ copies of the subpolyomino inside the dotted box. For $m = 22k + 13$ pixels, the number of pixel-guards is $6k + 4 = \frac{3m+5}{11}$, which, by elementary calculation, equals $\lfloor \frac{3m}{11} \rfloor + 1$.

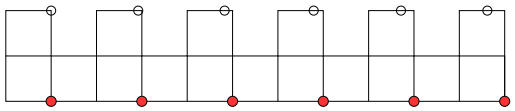


Figure 3.3: An example demonstrating that $\lfloor \frac{m+1}{3} \rfloor$ guards (depicted as filled dots) are sometimes necessary to guard P, since no two of the open circles can be covered by one guard.

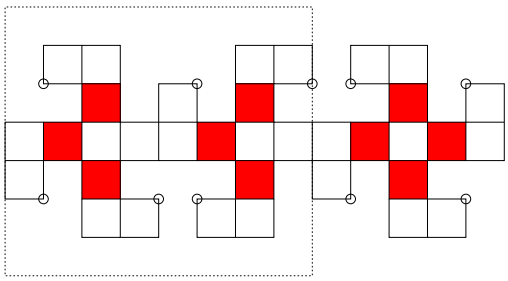


Figure 3.4: An example demonstrating that $\frac{3m+5}{11}$ pixel-guards (depicted as filled squares) are sometimes necessary to guard P, since no two of the circled corners can be covered by one pixel-guard.

3.2.2 Sufficiency Conditions

Point Guards

Given an m-polyomino P (possibly with holes) where $m \geq 2$, we show that $\lfloor \frac{m+1}{3} \rfloor$ guards are always sufficient to cover P. The proof of this sufficiency condition is obtained by arguing that we can iteratively remove certain types of *subpolyominoes* while leaving the remaining polyomino connected. The removal process will involve creating a BFS tree of the dual graph of P, and then iteratively clipping off small subtrees. First, we introduce a lemma to be employed later:

Lemma 3.2.1. *For $1 \leq m \leq 4$, any m-polyomino can be covered with one guard.*

Any 5-polyomino that is not $5'$ or $5''$ (see Figure 3.6) can be covered with one guard.

For $m = 6, 7$, any m-polyomino can be covered with two guards.

In all cases, the polyominoes are covered even in the r-visibility model.

Proof. One could prove this simply by inspection of the finite (though large) number of such polyominoes, but instead we give a proof based on locating the best places for guards using a spanning tree in the dual graph.

We do not prove each time that all of our guards are at pixel corners and that r-visibility suffices for the covering; this should be obvious from the constructions.

The crucial insight is that if a guard v is at the corner of some pixel p, then it guards not only p, but also all neighbors of p, even in the r-visibility model.

Lemma 3.2.2. *Any m-polyomino P, $m = 1, 2, 3, 4$, can be covered with one guard.*

Proof. If P is a rectangle, then any pixel corner will do. Otherwise P has a reflex corner. Placing a guard v at this corner means that v belongs to three pixels and covers the fourth (if any) since it is adjacent to one of the three. See Figure 3.5(a). □

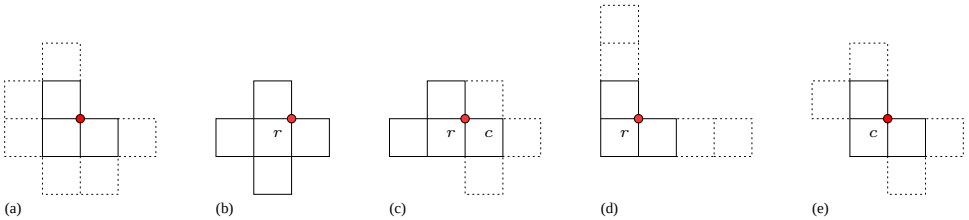


Figure 3.5: Cases for covering polyominoes with up to 5 pixels.

Lemma 3.2.3. *Any 5-polyomino P that is not $5'$ or $5''$ can be covered with one guard.*

Proof. Compute a BFS tree T of the dual graph of P, rooted at a node r of maximum degree. If r has degree 4, then any other pixel is a neighbor of r, so a guard on one corner of r will do (Figure 3.5(b)). If r has degree 3, then exactly one neighbor c of r has exactly one other neighbor (in T), and placing a guard at a common point of r and c will cover everything (Figure 3.5(c)).

Now assume that r has degree 2, hence the dual of P is a path. If P is a rectangle, then placing a guard at any pixel corner will do. If P has exactly one reflex corner, then placing a guard on it will cover everything (Figure 3.5(d)). If it has multiple reflex corners, then let c be the middle node of this path. If the neighbors of c are not on opposite sides of c, then the point common to the neighbors covers everything (Figure 3.5(e)). This finally leaves the case where c and its two neighbors form a rectangle, but P has two reflex corners. This implies that P is $5'$ or $5''$. □

Lemma 3.2.4. *Every (6) and (7) can be covered with two guards.*

Proof. We first prove this for a 7-polyomino P. Compute a BFS tree T of the dual graph of P, starting at a node r of maximum degree. If any child c of r has a subtree T_c of size 3 or 4, then T_c and $T - T_c$ form two connected

subpolyominoes that can be covered with one guard each, and we are done. If all children of r have subtrees of size 2 or 1, then place two guards at two diagonally opposite corners of r; these guards are then in all children of r and hence cover all grandchildren as well, and hence all of T.

So finally presume some child of r has a subtree of size 5 or 6. Then r can have at most two children, so the dual of P is a path. But then P can easily be split into a (3) and a (4) and hence be covered with two guards.

This proves the claim for a (7). For a 6-polyomino P, let q be a bottommost pixel, i.e., a pixel with smallest y-coordinate, breaking ties arbitrarily. Attach an extra pixel p below q to create a 7-polyomino P'. Then cover P' with two guards. If either one of them is in a bottom corner of p, then we can move it to the top corner of p without decreasing coverage, since p is a leaf in the dual graph. Hence the two guards will also cover P. □

Hence, Lemmas 3.2.2-3.2.4 establish the validity of Lemma 3.2.1. □

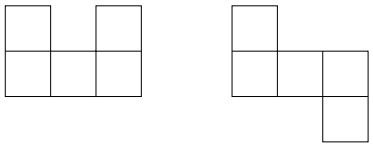


Figure 3.6: Special 5-polyominoes: $5'$ (left) and $5''$ (right). (The labels apply to all polyominoes that are one of the above after rotation and/or reflection.) Each requires two point guards.

In the proof of the subsequent theorem and again in Theorem 3.2.8, a BFS tree structure with an important construction property is utilized. Let T be a BFS tree of the dual graph of a polyomino P, rooted at a node r that has degree at most 2. Add the neighbors of r to the queue, with r as parent. As long as the queue is not empty, remove the next node v from it. Then, add all unvisited neighbors w of v to a queue in the following fashion: Assume that v has parent p_1. Then the neighbor w that is on the opposite side of v from p_1 is added *last* to the queue (if it exists and was not already visited.) In other words, the BFS will always give preference to "making a turn" when exploring the dual graph. Because of this, we have the following observation, which will be crucial later.

Figure 3.7: If the polyomino shown has a BFS tree rooted at p_2, v' cannot exist since otherwise c_1 would have been made a child of v', not v.

Lemma 3.2.5. *Let v be a pixel that has a grandparent in this BFS tree T. If v has two children c_1, c_2 in T that are on opposite sides of v, then v has no sibling.*

Proof. Let p_1 and p_2 be the parent and grandparent of v, respectively. Then p_2 must be adjacent to p_1, but it cannot be adjacent to either c_1 or c_2, since this would violate the BFS property. Since c_1 and c_2 are on opposite sides of v, this implies that p_2 and v are on on opposite sides of p_1. With our method of doing the BFS, therefore v is added last to the queue when exploring from p_1. If p_1 had any other child v', then v' would be adjacent to c_1 or c_2, and would have been added to the queue before v. So c_1 or c_2 would have been made a child of v', not v. Therefore, v has no siblings (see Figure 3.7). □

Theorem 3.2.6. *For an m-polyomino P_0 (possibly with holes) there are polyominoes $P_1, P_2, \ldots, P_f$ with the following properties:*

(a) P_i is a connected subpolyomino of P_{i-1} $(1 \leq i \leq f)$.
(b) Subpolyomino S_i, the difference between P_{i-1} and P_i is in the set Good Polyominoes,
$GP = \{(3), (4), ((5)\backslash\{5', 5''\}), (6), (7)\}$.
(c) P_f has 0, 1, or 2 pixels.

Proof. Let T be a BFS tree of the dual graph of P_0 obtained as explained above. Every node in T has at most 3 children. If the height of T is less than 2, then T has at most 3 nodes since T is rooted at a node of degree 2. If T has exactly 3 nodes, then set $S_f = T$ (i.e., let S_f be the subpolyomino whose dual graph is T), set $P_f = \emptyset$ and we are done. If $|T| < 3$, set $P_f = T$ and again we're done

Suppose now that the height of T is at least 2, and let q be a lowest leaf of T. If q has siblings, then let T_{p_1} be the subtree rooted at the parent p_1 of q. See Figure 3.8(a). T_{p_1} then has 3 or 4 nodes. Set $S_1 = T_{p_1}$, satisfying (b). Hence, $P_1 = P_0 - S_1$ is $T - T_{p_1}$; this is connected (so satisfies (a)) since T_{p_1} is a rooted subtree of T. By induction we can split P_1 as desired.

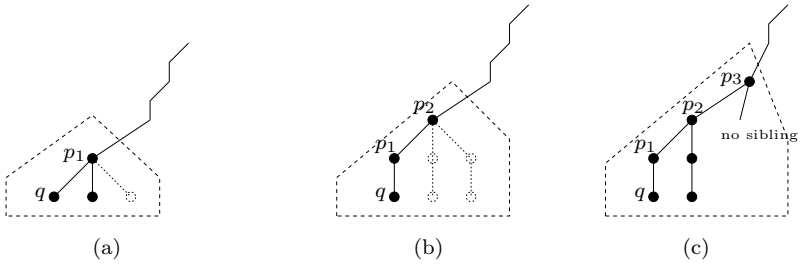


Figure 3.8: Finding a subtree that forms a Good Polyomino.

If none of the lowest leaves of T has siblings, let q be a lowest leaf and p_1, p_2 be its parent and grandparent. Let T_{p_2} be the subtree rooted at p_2. See Figure 3.8(b). T_{p_2} has between 3 and 7 nodes. The minimum of 3 occurs when p_2 has no other children and the maximum of 7 occurs when p_2 has 3 children, each of whom has one child (since no leaf has siblings). If T_{p_2} is not $5'$ or $5''$, then set $S_1 = T_{p_2}$. As before, $S_1 \in GP$, satisfying (b) and since $T - T_{p_2}$ is connected, (a) holds as well and we are done by induction.

If T_{p_2} is $5'$ or $5''$, then p_2 must be the middle pixel, and we do one of the following:

(1) If p_2 has no parent, then the whole polyomino P_0 is a $5'$ or $5''$, and hence can be split into a (3) and a (2).
(2) If p_2 has a grandparent, then p_2 has no sibling since it is the middle node of a $5'$ or $5''$ and hence has children on opposite sides (Lemma 3.2.5). Therefore, the subtree T_{p_3} rooted at the parent p_3 of p_2 is a (6). See Figure 3.8(c). Set $S_1 = T_{p_3}$ and iterate as before.
(3) Finally presume p_2 has a parent p_3, but no grandparent. So p_3 is the root and has degree ≤ 2. Let T' be the (6) formed by T_{p_2} together with p_3; set $S_1 = T'$ as before, then $P_1 = P_0 - S_1$ is again a connected polyomino and we are done by induction.

□

Corollary 3.2.7. *For $m \geq 2$, $\lfloor \frac{m+1}{3} \rfloor$ guards are sometimes necessary and always sufficient to cover a connected m-polyomino P (possibly with holes), even in the r-visibility model.*

Proof. Theorem 3.2.6 shows that we can partition P into subpolyominoes $S_1, \ldots, S_f$ and P_f such that each S_i is covered by $\lfloor \frac{|S_i|}{3} \rfloor$ point guards, and P_f has 0, 1 or 2 pixels. We use $1 = \lfloor \frac{|P_f|+2}{3} \rfloor$ point guard for P_f if it is non-empty. These guards cover the polyomino even in the r-visibility model by Lemma 3.2.1. Hence, the number of point guards is $\sum_{i=1}^{f} \lfloor \frac{|S_i|}{3} \rfloor + \lfloor \frac{|P_f|+2}{3} \rfloor \leq \frac{m+2}{3}$. Since the number of guards is an integer, this gives an $\lfloor \frac{m+2}{3} \rfloor$ sufficiency condition, but we can make a slight improvement to $\lfloor \frac{m+1}{3} \rfloor$.

If $m = 2 + 3k$ or $m = 3 + 3k$ with $k \in \mathbb{N}$, then $\lfloor \frac{m+1}{3} \rfloor$ and $\lfloor \frac{m+2}{3} \rfloor$ are equivalent. So assume $m = 1 + 3k$.

Suppose first that some S_j is not a (3) or a (6), and hence uses actually only $\lfloor \frac{|S_j|-1}{3} \rfloor$ guards. Re-doing the above equation then shows that the number of guards is at most $\frac{m+1}{3}$ (and as before, therefore not more than $\lfloor \frac{m+1}{3} \rfloor$ since it is an integer.) On the other hand, if each S_i is a (3) or a (6), and $m = 1+3k$, then P_f consists of exactly one pixel. Hence, $S_f \cup P_f$ is a (4) or a (7), and can be covered with $\lfloor \frac{1}{3}(|S_f| + |P_f| - 1) \rfloor$ point guards. Again re-doing the equation shows that the number of guards is at most $\lfloor \frac{m+1}{3} \rfloor$. □

Note that our proof is constructive and gives an algorithm to find a set of $\lfloor \frac{m+1}{3} \rfloor$ guards. The time complexity of this algorithm is dominated by the time to find the decomposition of P into good subpolyominoes. To see that this can be done in overall linear time, observe that there is no need to re-compute the BFS tree every time: the remainder of the BFS tree is a BFS tree for the remaining polyomino. Thus we compute the BFS tree only once, and enumerate the nodes in it in backward level order. For each node q, in this order, we then find a good subpolyomino in the vicinity of q: it is either at q's parent, grandparent, or the great-grandparent, or at one of their children or grandchildren. Thus we only need to check a constant number of subtrees, all within constant distance of q. So finding a good subpolyomino takes constant time per removed subpolyomino.

Pixel Guards

We proved above that $\lfloor \frac{m+1}{3} \rfloor$ point guards are sufficient to guard any m-polyomino. Pixel guards are more powerful than point guards, and, thus, one would expect that fewer pixel guards are needed. Indeed, as stated in Theorem 3.2.8 below, the sufficiency condition for pixel guards agrees with the lower bound $\lfloor \frac{3m}{11} \rfloor + 1$. As in the point guard case, the sufficiency proof involves removing subtrees of a BFS tree of the dual of a polyomino. The proof, which involves an extensive case analysis and many technical details, can be found along with other related lemmas in Appendix A.

The algorithm to find these guards is similar to that for point guards: Compute a BFS tree, process nodes in bottom-up order, and, for each q, find a suitable subpolyomino via some subtree that has constant distance from q. The algorithm takes linear time.

Theorem 3.2.8. $\lfloor \frac{3m}{11} \rfloor + 1$ *pixel guards are sufficient to cover an m-polyomino P (possibly with holes). The same bound holds in the r-visibility model.*

We note that the 'power' of a pixel guard is due to it being closed (including the boundary of the pixel). For open pixel guards, the effectiveness is dramatically weakened. Figure 3.9 shows that $\lfloor \frac{m}{2} \rfloor$ open pixel guards are sometimes necessary. Also, $\lfloor \frac{m}{2} \rfloor$ open pixel guards are always sufficient: overlay a chessboard with black and white squares on P and place open pixel guards on the squares with color (black or white) that overlap with P the least. We then have the following result:

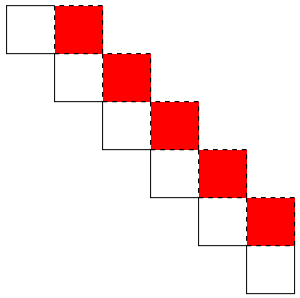


Figure 3.9: $\lfloor \frac{m}{2} \rfloor$ open pixel guards are sometimes necessary to cover a polyomino.

Theorem 3.2.9. $\left\lfloor \frac{m}{2} \right\rfloor$ *open pixel guards are sometimes necessary and always sufficient to cover an* m*-polyomino* P *(possibly with holes). The same bound holds in the* r*-visibility model.*

3.2.3 Generalization to Rectanglominoes

We observe that the combinatorial results described above extend to a generalization of polyominoes, which we will call *rectanglominoes*: connected unions of m edge aligned rectangular pixels. If pixels p_i and p_j, with heights h_i and h_j and lengths l_i and l_j respectively, are horizontally adjacent in a rectanglomino R, then $h_i = h_j$. Similarly, if p_i and p_j are vertically adjacent, then $l_i = l_j$. Any rectanglomino R has an *associated* polyomino P_R that is obtained by setting all pixel heights and lengths of R to unit length. It is possible that P_R will self-overlap, but the combinatorial results obtained for polyominoes apply to self-overlapping polyominoes as well. Clearly, the dual graphs of R and P_R are equivalent since the number of pixels in R and P_R is the same and pixel adjacencies are preserved.

Despite having identical dual graphs, a rectanglomino and its associated polyomino do not always have the same guard number. Consider the example in Figure 3.10 where the rectanglomino on the left requires three point guards while the associated polyomino on the right needs only two.

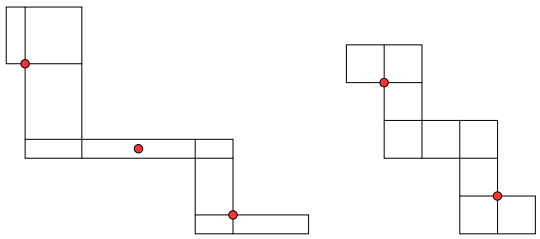


Figure 3.10: The rectanglomino (left) requires 3 point guards, while its associated polyomino (right) requires only 2 point guards.

However, the two numbers are the same in the r-visibility model:

Lemma 3.2.10. *A rectanglomino* R *can be covered with* k *guards in the* r*-visibility model if and only if its associated polyomino* P_R *(possibly self-overlapping) can be covered with* k *guards in the* r*-visibility model.*

Proof. We only show one direction; the other one is similar. Assume we have a cover of P_R. Map each point p_r in P_R to a point r in R in the natural way: if p_r is at a corner, then it is mapped to the corresponding corner, and if it is not on a corner, then it is mapped to the linear interpolation between the corners of the pixel that contain it. Using this mapping on the set of guards of P_R gives a set of points in R, and we must now argue that this is a cover of R.

Since adjacent rectangles in a rectanglomino are edge-aligned, this transformation maps a rectangle in P_R to a rectangle in R. Hence, if point p_R in P_R is guarded in the r-visibility model by guard v_R, then the rectangle $\mathcal{R}$ spanned by them is inside P_R. Applying the transformation yields a rectangle inside R containing the images of p_R and v_R; hence, any point in R is covered by some of the chosen guards. □

As a consequence, all of our upper bounds on the guard number (Corollary 3.2.7 for point guards and Theorem 3.2.8 for pixel guards) immediately transfer to rectangle*minoes*, since our covers were valid even in the r-visibility model. Of course the lower bounds transfer as well, since a polyomino is a special kind of rectanglomino. Hence, we have:

Corollary 3.2.11. $\lfloor \frac{m+1}{3} \rfloor$ *point guards are always sufficient and sometimes necessary to cover an* m*-rectanglomino (possibly with holes). Also,* $\left\lfloor \frac{3m}{11} \right\rfloor + 1$ *pixel guards are always sufficient and sometimes necessary to cover an* m*-rectanglomino.*

3.3 Hardness

Theorem 3.3.1. *Given a simple* m*-polyomino* P *and an integer* k*, it is NP-hard to determine whether* k *pixel guards are sufficient to cover* P*. The same statement holds for point guards, and for both guard types in the all-or-nothing visibility model.*

Proof. We first give an overview. The result is shown using a reduction from MAX2SAT(2L), which is the special case of MAX2SAT having each literal appear at most twice. Brodén et al. [13] gave a reduction from this problem to the Minimum Line Covering Problem (MLCP): For a given set L of lines in the plane, find a minimum cardinality point set S such that each line in L

has at least one point of S on it. Their reduction (intentionally) used lines such that none are parallel. We do the same reduction, but modify the gadgets such that all lines are in the octagonal grid, i.e., they are all horizontal, vertical or have slope ± 1. (We also correct a small error in their APX-hardness proof.) We can also show that all intersection points are within a polynomial-sized box. Thus we show that MLCP remains NP-hard and APX-hard even with lines in the octagonal grid.

We then surround these lines with a rectangle, with further attachments to enforce that guarding the polyomino with few guards is feasible only by placing guards roughly where the lines used to be. Consequently we have a polyomino that mimics the line arrangement of MLCP and hence proves NP-hardness. We were unable to create a reduction that allows us to prove APX-hardness; we give some indications as to where the difficulties were.

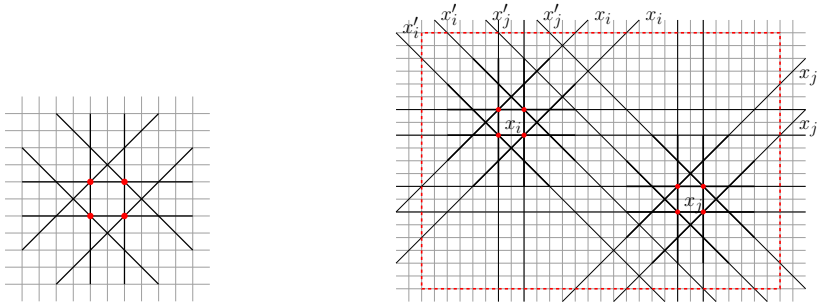
3.3.1 From MAX2SAT(2L) to MLCP in octagonal grids

Let Q be an instance of MAX2SAT(2L) with variables U and clauses C. Brodén at al. [13] create out of Q a line arrangement with arbitrary slopes such that at most 3 lines meet in a point. We use the exact same idea for gadgets to have lines in the octagonal grid; we review the ideas here to keep the presentation independent.

Variable gadget. A *variable gadget* consists of 8 lines as in Figure 3.11a. It has 4 intersection points of 3 lines. We place the lines to make all intersections be on grid points. The two diagonals represent the positive literal and the two off-diagonals the negative literals of the variable. These are referred to as *literal lines*. Literal lines will later be connected to lines of clause gadgets according to their participating variables.

Two variable gadgets are placed as in Figure 3.11b. The second variable is positioned carefully such that no new intersection of 3 lines is created. We are able to place $|U|$ variable gadgets in the same manner.

Nesting boxes. Let Box_0 be the box containing all intersection points of lines derived from variable gadgets. The width w_0 and height h_0 of Box_0 are linear in the number of variables. We now define $|C|$ nested boxes $\text{Box}_0 \subset \text{Box}_1 \subset \cdots \subset \text{Box}_{|C|}$, one per clause. For $k \geq 1$, box Box_k is made so much bigger than Box_{k-1} that its top intersects all literal lines, yet these intersection points are all outside the horizontal range spanned by Box_{k-1}. See Figure 3.12



(a) Single gadget with 4 intersection points of 3 lines

(b) Placement of two gadgets

Figure 3.11: Variable gadget

for an example with $|C| = 3$. One can see that it suffices for box Box_k to have width $(2k-1)w_0 + 2kh_0$ and height $2kw_0 + (2k+1)h_0$, therefore, the largest box $\text{Box}_{|C|}$ has width and height $O(|C||U|)$.

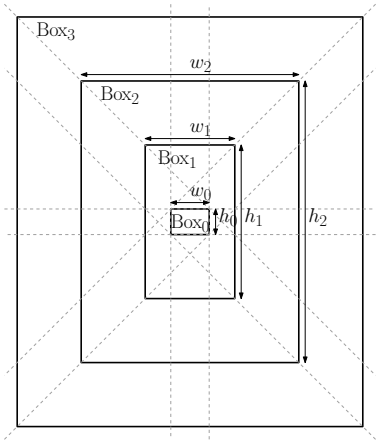


Figure 3.12: Nesting boxes used to place clause gadgets

Clause gadget. One *clause gadget* consists of 3 lines—1 horizontal and 2 vertical. Figure 3.13 provides examples. Clause gadgets have 2 intersection points of 3 lines, each representing a participating literal of the clause.

For the k-th clause, we place its horizontal line on the top boundary of Box_k. Then we place two vertical lines so that they create an intersection of three lines with the diagonal or off-diagonal literal lines of the literals of the clause. There are two possible literal lines to select from for each literal.

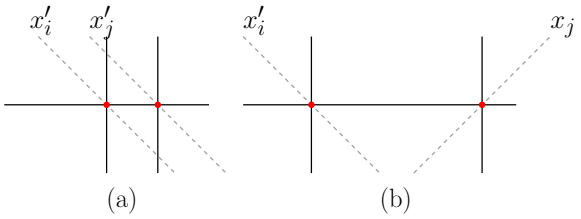


Figure 3.13: (a) clause gadget for two negated variables; (b) clause gadget for a variable and a negated variable

By construction of Box_k, the vertical lines are outside Box_{k-1} but intersect Box_k. Hence the number of intersections of 3 lines introduced by each clause is exactly 2. All other intersections are those of 2 lines.

Combination of variable and clause gadgets. Since we have equivalent gadgets as Brodén et al.'s [13], the rest of the argument is the same. We have a line arrangement with $8|U| + 3|C|$ lines, and want to cover the lines with as few points as possible. Exactly as in [13], one shows that the instance of MAX2SAT(2L) can have c clauses satisfied if and only if the instance of MLCP can be covered using $k = \frac{1}{2}(6|U| + 3|C| - c)$ points. In particular, if c^* and k^* are the optimal solutions, then $c^* = 6|U| + 3|C| - 2k^*$ (and vice versa $k^* = \frac{1}{2}(6|U| + 3|C| - c^*)$. This proves NP-hardness of MLCP in octagonal grids.

We now briefly review the APX-hardness proof and correct a small error in Brodén et al.'s [13]. It is known [7] that it is hard to approximate MAS2SAT(2L) within a factor of $\frac{2012}{2011} - \varepsilon$ for any $\varepsilon > 0$. To avoid having to deal with arbitrary ϵ (and because the constants will be extremely close to 1 anyway), we will use $\varepsilon = \frac{1}{4022}$, and it is hence hard to approximate MAX2SAT(2L) within a factor of $\frac{4023}{4022}$. In particular, if c_A is the number of satisfied clauses obtained with some polynomial-time algorithm, then $c_A < \frac{4022}{4023}c^*$.

Now assume there exists an f-approximation algorithm for MCLP, so it finds in polynomial time a set of $k_A \leq f \cdot k^*$ points that cover all lines. Using the relationship between the instances, we can therefore find $c_A = 6|U| + 3|C| - 2k_A$ satisfied clauses. Putting it all together, we get

$$f \geq \frac{k_A}{k^*} = \frac{\frac{1}{2}(6|U| + 3|C| - c_A)}{\frac{1}{2}(6|U| + 3|C| - c^*)} > \frac{6|U| + 3|C| - \frac{4023}{4022}c^*}{6|U| + 3|C| - c^*}$$
$$= 1 + \frac{1}{4023} \cdot \frac{c^*}{6|U| + 3|C| - c^*}.$$

To bound this, Brodén et al. use two inequalities: $|U| \leq 2|C|$ and $c^* \leq |C|$. Both are correct, but the second one goes in the wrong direction to be applied. Instead we use $c^* \geq |C|/2$ (we can always satisfy half of the clauses by using an arbitrary variable assignment or its opposites). Therefore

$$\begin{aligned} f \geq 1 + \frac{1}{4023} \cdot \frac{c^*}{6|U| + 3|C| - c^*} &\geq 1 + \frac{1}{4023} \cdot \frac{|C|/2}{12|C| + 3|C| - |C|/2} \\ &= 1 + \frac{1}{4023 \cdot 29} = \frac{116668}{116667}. \end{aligned}$$

This proves APX-hardness of MCLP, even with lines in the octagonal grid.

3.3.2 From MLCP to Polyominoes

We now proceed to the next step: convert the line arrangement into a polyomino such that guarding the polyomino with guards corresponds to covering the lines. Assume all intersection points of the instance of MLCP were inside a rectangle of size $\ell \times h$ (which was polynomially bounded.) The main part of the polyomino consists of this rectangle (which we will call the *main rectangle*), with so-called *ray gadgets* attached which emit a virtual ray that mimics a line of the line arrangement and must contain a guard. Figure 3.16 outlines the whole construction. Since we have vertical, horizontal and diagonal lines, two types of ray gadgets are required.

Ray gadget. Figure 3.14 shows the gadget to generate a horizontal ray. The bold black solid line is the boundary of the polyomino. In order to see any portion of the open segment $\overline{ab}$ of the spike, a pixel guard must exist on or above $\overrightarrow{ac}$ and on or below $\overrightarrow{bd}$. Some possible pixel guards are shown as squares in the figure. If we set the length of the spike to be the same as the width ℓ of the main rectangle, then possible guard locations outside of the spike for $\overline{ab}$ form a 3 by ℓ horizontal strip. The ray gadget for a vertical ray is symmetric.

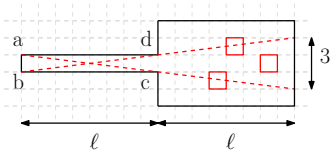


Figure 3.14: A gadget for a horizontal ray

The other type of gadget generates slant-rays as in Figure 3.15. The gadget has lots of legs (paths of length 3 that end in a corner), and we need at least

one guard to cover this portion. The best placement is in the third pixel from the end. This covers almost everything of the gadget, even in the all-or-nothing model, but (part of) one pixel near the bottom is not covered. The uncovered part is indicated in black in Figure 3.15. We call this the *bottom region.* In order to guard this bottom region, we need at least one more pixel guard, which is either inside the gadget or along the narrow wedge defined by the bottom region and the reflex corners near the connection of the gadget to the main rectangle. This wedge resembles a ray of slope 2.

We choose the height of the gadget to be the same as the height of main polyomino; then all pixels that intersect the wedge are either on the ray, or adjacent to a pixel on the ray that the wedge mimics. We can similarly build a gadget for a ray that has slope (roughly) -2.

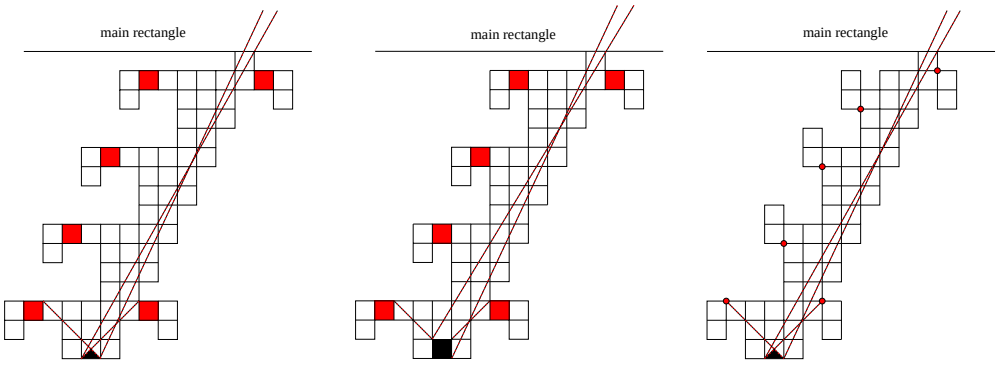


Figure 3.15: A gadget for a slant-ray. We show the gadget for pixel guards (the same gadget works for the unrestricted or the all-or-nothing model, but the bottom region is different), and for point-guards in the unrestricted model.

Refinement of the grid. Let $\mathcal{L}$ be the line arrangement with horizontal, vertical and diagonal lines, i.e., the instance of MLCP. All lines are part of an octagonal grid G. We refine the grid by repeatedly doubling all lines until all grid points that contain lines from $\mathcal{L}$ are at least 11 units apart horizontally. Then we skew the drawing to double its height without changing its width. Then all lines are now horizontal, vertical or of slope ± 2. Note that by construction any two lines of the line arrangement intersect in a grid point.

Constructing the polyomino. Let B be the box that contains all intersection points of all lines after this transformation. Now define the polyomino by adding all pixels within B. Furthermore, for every horizontal or vertical line in $\mathcal{L}$, we attach a gadget for a horizontal/vertical ray where the line hits

the left/bottom side of B. For any line of slope ± 2, we attach the gadget for a slant-ray where the line hits the bottom of B. Since we refined the grid, gadgets for two lines of the same slope are sufficiently far apart so that they do not interfere. Since all intersection points are inside B, the lines are sorted by slope around the boundary of B and so gadgets of lines of different slopes will not intersect each other. Further, if we make gadgets long enough then no forced guard in one gadget can be used to cover the unguarded regions in other gadgets and the wedges defined by the gadgets are disjoint except where corresponding lines intersect. Using gadgets of length l or h will suffice; this will hence at most triple the height/width of the grid containing the construction. Hence the polyomino has width and height $O(lh)$.

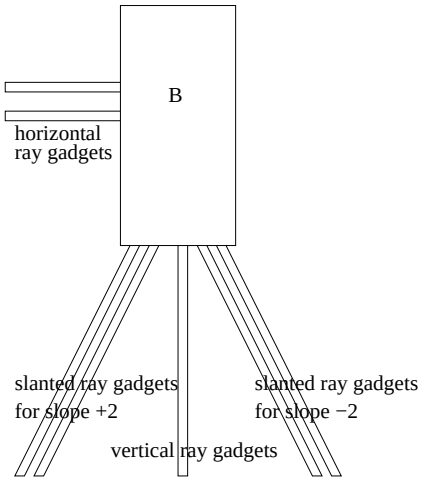


Figure 3.16: Place the ray gadgets around the box that contains all intersections. The width of the gadgets is not to scale.

Summary. It is now easy to see that a minimum line cover set S of k points corresponds exactly to a placement of $N + k$ pixel guards that cover everything. Here N is the number of guards required for the gadgets for slant-rays; this number is known a-priori and polynomial in the number of lines and the width and height of the box bounding the instance of MLCP. Hence guarding the polyomino with the minimum number of pixel guards corresponds to finding a minimal line cover set. This proves NP-hardness of guarding a simple polyomino with pixel-guards, both in the unrestricted model and the all-or-nothing model. The proof for point-guards is quite similar, using the modified slant-ray gadget. □

Remark. We note that our reduction does not also show APX-hardness as in [13]. This is due to the N guards required for the slant-ray gadgets, which can be large and consequently give us no guarantee of an inapproximability ratio greater than one. APX-hardness would hold if we could construct a slant-ray gadget that uses only a constant number of additional guards (because then $N \leq const \cdot k^*$ since every slant line requires at least one guard.) It is not difficult to construct such a gadget for one ray (using a rectangle with just two attachments), but we have not been able to construct such a gadget that is "slim enough" that we can place them next to each other for all slant-ray gadgets without overlapping.

3.4 Algorithms for Special Cases

Since guarding polyominoes is NP-hard, we consider special cases for which one can compute the optimum in polynomial time. In particular, consider a *thin polyomino tree*, which is a polyomino for which the dual graph is a tree. The tree structure makes it plausible that a good set of guards can be found easily, and we show this now.

The algorithms that we found provably work in both the all-or-nothing model and the r-visibility model; as we will see, these two concepts are the same in a thin polyomino tree. The reason that these models are easier lies in the following lemma, which demonstrates that there are only finitely many point guard placement positions to consider, rendering the problem a set cover problem (and hence immediately yielding $O(\log m)$ approximation algorithms.)

Lemma 3.4.1. *In the r-visibility model, in any polyomino P there exists an optimal solution for guarding P such that the guards are placed only at pixel corners.*

Proof. Move any guard g not placed on a corner to one of the corners, g', of the pixel that contains it. To see that this is still a cover, consider a point p that was guarded by g. Then the rectangle R spanned by p and g is inside P. Round out R to the rectangle R' consisting of all pixels of P intersected by R; then, $R' \subseteq P$. But R' contains both g' and p, and hence they are visible to each other in the r-visibility model. □

Lemma 3.4.2. *In a thin polyomino tree P, a point q is r-visible from a point p if and only if q is all-or-nothing visible from p.*

Proof. Suppose q is all-or-nothing visible from p, and let q' be the pixel containing q and completely seen by p. Then the four sight lines from p to the corners of q' are all within P. If p is not within horizontal or vertical range of q, then these four sight lines cross at least two different neighbors of q'. Following the sight lines, we hence walk between these neighbors of q', using only pixels inside P, without using pixel q'. This is impossible if the dual graph is a tree.

The other direction is straightforward using Lemma 3.4.1 and holds even if P is not a thin polyomino tree. Let R be the rectangle spanned by p and q, and expand it to the union of polyominoes R' as in that proof. Since R was in P, so must be R'. Note that R' contains both p and the pixel containing q, and so p guards q in the all-or-nothing model. □

We note here that these lemmas are not valid in the general model (unrestricted visibility).

3.4.1 Guarding Thin Polyomino Trees

Consider an arbitrary thin polyomino tree P. By Lemma 3.4.2, r-visibility is equivalent to all-or-nothing visibility in P. From [50], we can obtain in polynomial time a minimum cardinality r-star cover. Hence, by placing a point guard in each r-star polygon found in the cover, we obtain an optimal solution in the all-or-nothing visibility model.

The algorithm to find this cover is polynomial, but very slow. It remains open to devise an algorithm for optimal cover in thin polyomino trees that has quadratic (or perhaps even linear) runtime. We also leave as an open question as to whether the optimal number of point guards can be found in a thin polyomino tree in the general visibility model.

3.4.2 Guarding Thin Polyomino Paths

For the special case of a *thin polyomino path* P, we give simple greedy algorithms for placing the minimum number of point or pixel guards in P along with proofs of optimality. The pixels of degree 2 in a thin polyomino path come

in two varieties: *bridge* pixels, which have neighbors on two opposite sides, and *corner* pixels, which have neighbors on two adjacent sides. The pseudocode and proof of optimality for both the point and pixel guard versions are given below.

Pseudocode

Algorithm: `ThinPolyominoPathPointGuardCover`(P)

while there are pixels remaining in P
- **1.** Begin at a pixel with degree = 1. Set corners = 0.
- **2.** **while** the next pixel (if it exists) is a bridge pixel
  - Append the pixel to S.
- **3.** **while** the next pixel (if it exists) is a corner pixel and corners < 3
  - Append the pixel to S.
  - corners = corners + 1.
- **4.** **while** the next pixel (if it exists) is a bridge pixel
  - Append the pixel to S.
- **5.** **if** the next pixel exists
  - Append the pixel to S.
  - Remove the star-shaped thin subpolyomino S just traversed.
  - Place a point guard at a vertex that covers S.

Algorithm: `ThinPolyominoPathPixelGuardCover`(P)

while there are more pixels remaining in P
1. Initialize S to be the the pixel with degree = 1 just created or else any leaf pixel.
Set corners = 0 and bridges = 0.
2. **while** the next pixel (if it exists) is a bridge pixel
Append the pixel to S.
3. **while** the next pixel (if it exists) is a corner pixel and corners < 5
Append the pixel to S.
corners = corners + 1.
4. **while** the next pixel (if it exists) is a bridge pixel
Append the pixel to S.
bridges = bridges + 1.
5. **if** bridges = 1 and corners ≤ 2
set corners = 0.
while the next pixel (if it exists) is a corner pixel and corners < 2
Append the pixel to S.
corners = corners + 1.
6. **while** the next pixel (if it exists) is a bridge pixel
Append the pixel to S.
7. **if** the next pixel exists
Append the pixel to S.
Remove the star-shaped thin subpolyomino S just traversed.
Place a pixel guard inside S that covers S.

3.4.3 Proofs of Optimality

Proof of Optimality for `ThinPolyominoPathPointGuardCover`

Lemma 3.4.3. *Each subpolyomino partition created by* `ThinPolyominoPathPointGuardCover` *is star-shaped.*

Proof. Consider one iteration of the main while loop. Step **2.** of the algorithm appends pixels along a straight path. By the end of step **2.** the subpolyomino is comprised only of bridge pixels. During step **3.**, up to three corner pixels are appended in succession. If no corner pixels are appended however, then the subpolyomino is a straight path and thus trivially star-shaped. Otherwise, if we put a point guard at the reflex vertex of the first corner pixel appended that is furthest from the pixel we started at, then we cover the bridge pixel(s) appended from step **2.**, the corner pixels of step **3.**, all subsequent bridge pixels and possibly one corner pixel appended in steps **4.** and **5.** (see Figure 3.17). Thus, each subpolyomino partition is star-shaped. □

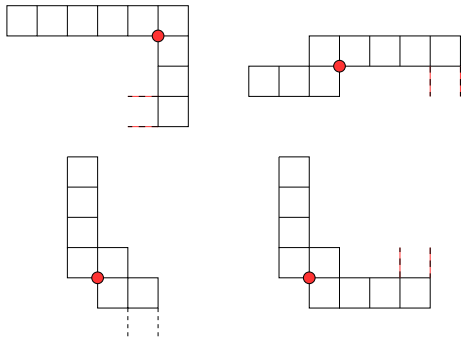


Figure 3.17: The four free subpolyomino types that can be constructed in one iteration of the main while loop of `ThinPolyominoPathPointGuardCover`, depending on whether there are 1, 2, or 3 corner pixels in the subpolyomino (the last of which has two subcases shown in the bottom row). By placing a point guard at the reflex vertex of the first corner pixel appended that is furthest from the first pixel encountered as shown for each case, the entire subpolyomino is covered.

Lemma 3.4.4. *During an iteration of the main while loop, the algorithm* `ThinPolyominoPathPointGuardCover` *appends a maximal number of pixels to the star-shaped subpolyomino partition constructed.*

Proof. Suppose the claim is false: then another pixel could have been appended at the end of the subpolyomino partition constructed while remaining star-shaped. If there was another pixel to append, then the last pixel we appended was a corner pixel in the original pixel path according to the algorithm. Therefore, attaching a subsequent pixel would result in either 4 corner pixels appended in succession or two corner pixels separated by one or more bridge pixels. Neither resulting subpolyomino can be covered with one point guard. Hence, the number of pixels in each star-shaped subpolyomino partition constructed by the algorithm is maximal. □

Theorem 3.4.5. *After the termination of* `ThinPolyominoPathPointGuardCover`*, each subpolyomino partition contains an independent witness point at the center of the first pixel encountered by the algorithm.*

Proof. If P is star-shaped, then we have just one subpolyomino partition, which trivially has an independent witness point at the center of the first pixel.

Consider the case when we have $k > 1$ subpolyomino partitions. We see that no point exists in the set of pixels along the path from the first pixel of the first subpolyomino to the first pixel of the second subpolyomino that can cover the first pixels of both the first and second subpolyominoes in their entirety. Otherwise, this first subpolyomino was not maximal in size. Therefore, we can place an independent witness point at the center of the first pixel of the first subpolyomino.

If there is a third subpolyomino partition, we observe similarly that no point can exist in the set of pixels along the path from the first pixel of the second subpolyomino to the first pixel of the third polyomino that covers the first pixels of both the second and third subpolyominoes. This implies that we can place an independent witness point at the center of the first pixel.

Continuing this line of reasoning, we observe that each of the first $k-1$ subpolyomino partitions has an independent witness point at the center of its first pixel. There is also no point along the path from the first pixel of the $(k-1)^{th}$ subpolyomino to the first pixel of the k^{th} subpolyomino that covers the first pixels of both the $(k-1)^{th}$ and k^{th} subpolyominoes. Hence, the $(k-1)^{th}$ and k^{th} subpolyominoes each have an independent witness point at the center of their first pixels, respectively. □

Corollary 3.4.6. `ThinPolyominoPathPointGuardCover` *provides an optimal solution for any polyomino path P in the all-or-nothing visibility model.*

Proof. Suppose the algorithm gives us k subpolyominoes in the partition so that k point guards are used. Since each subpolyomino has an independent witness point by Theorem 3.4.5, $k \leq OPT$. Clearly, k point guards is sufficient to cover P. Hence, $OPT = k$. □

Proof of Optimality for `ThinPolyominoPathPixelGuardCover`

Lemma 3.4.7. *Each subpolyomino partition created by* `ThinPolyominoPath-PixelGuardCover` *is star-shaped.*

Proof. Consider one iteration of the main while loop. Step **2.** of the algorithm appends pixels along a straight path. By the end of step **2.** the subpolyomino is comprised only of bridge pixels, forming a rectangle, which is star-shaped. During step **3.**, up to 5 corner pixels can be appended. If 3, 4 or 5 corner pixels are appended, then we get a subpolyomino of type c, d, or e as depicted

in Figure 3.18. All subpolyomino types in Figure 3.18 can be covered with just one pixel guard.

If no corner pixels are appended, then the subpolyomino is a straight path and is trivially star-shaped.

If one corner pixel is appended in step **3**, then we must consider what happens in step **4**: if only one bridge pixel is appended, then we get either subpolyomino type f or g depending on whether one or two corner pixels are subsequently attached in step **5.** If, instead, two or more bridge pixels are appended in step **4.**, then step **5.** is skipped and we obtain subpolyomino type a.

Now, if two corner pixels are appended in **3.**, then we must again consider what happens in step **4**: if only one bridge pixel is appended, then we get either subpolyomino type h or i depending on whether one or two corner pixels are subsequently attached in step **5.** If instead two or more bridge pixels are appended in step **4.**, then step **5.** is skipped and we obtain subpolyomino type b.

Therefore we see by case analysis that every subpolyomino type produced by one iteration of `ThinPolyominoPathPixelGuardCover` is star-shaped. □

Lemma 3.4.8. *During an iteration of the main while loop,* `ThinPolyomino-PathPixelGuardCover` *appends a maximal number of pixels to the star-shaped subpolyomino partition constructed.*

Proof. Suppose the claim is false: then another pixel could have been appended at the end of the subpolyomino partition constructed while remaining star-shaped. If there was another pixel to append, then the last pixel we appended was a corner pixel in the original pixel path according to the algorithm. Therefore, attaching a subsequent pixel would result in one of the following pixel sequences in the subpolyomino by exhaustion (where C stands for corner pixel, B stands for bridge pixel and, '...' implies possibly more of the preceding pixel type):

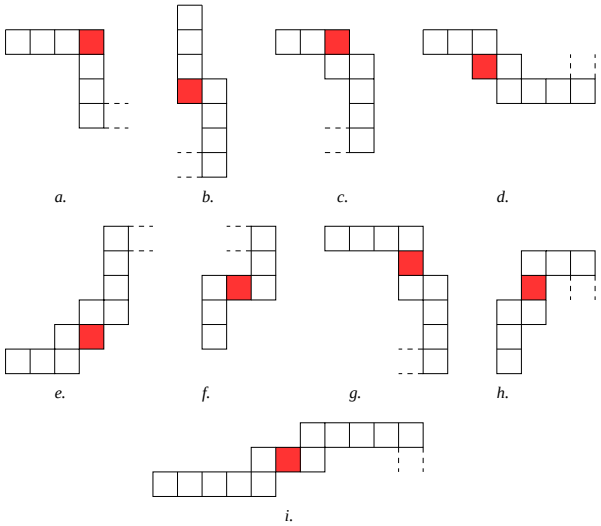


Figure 3.18: The nine possible free subpolyomino types that can be constructed from one iteration of the main while loop in `ThinPolyominoPathPixelGuardCover`. Each subpolyomino type is coverable with one pixel guard as shown with the filled (solid) pixels.

1. C, C, C, C, C, C
2. C, B, ..., B, C
3. C, B, C, C, C
4. C, C, C, B, C

None of these pixel sequences can be covered with one pixel guard. Hence, the number of pixels in each star-shaped subpolyomino partition constructed by the algorithm is maximal. □

Theorem 3.4.9. *After the termination of* `ThinPolyominoPathPixelGuardCover`*, each subpolyomino partition contains an independent witness point at the center of the first pixel encountered by the algorithm.*

Proof. The proof is analogous to Theorem 3.4.5. □

Corollary 3.4.10. `ThinPolyominoPathPixelGuardCover` *provides an optimal solution for any polyomino path P in the all-or-nothing visibility model.*

Proof. The proof is analogous to Corollary 3.4.6. □

3.5 Conclusion

We have explored a variation of the art gallery problem set in a polyomino domain, in which the input parameter, m, denotes the number of pixels found in the polyomino as opposed to the usual parameter, n, the number of vertices of the polygon. Using point guards, it was shown that $\lfloor \frac{m+1}{3} \rfloor$ guards are sometimes necessary and always sufficient to cover a polyomino on m pixels. In the case of pixel guards, we have that $\lfloor \frac{3m}{11} \rfloor + 1$ guards are sometimes necessary and always sufficient. When the polyomino is generalized to a rectanglomino, we demonstrated that these bounds were preserved.

It should be emphasized that these conditions apply to polyominoes with or without holes. For general polygons, $\lfloor \frac{n+h}{3} \rfloor$ point guards are sometimes necessary and always sufficient where h is the number of holes [11]. For $m \leq \frac{3n}{4} - 4$ we have $\lfloor \frac{m+1}{3} \rfloor < \lfloor \frac{n}{4} \rfloor$, yielding a strictly lower sufficiency bound for point guards than obtained by the art gallery theorem for orthogonal polygons [29].

We have shown that guarding an m-polyomino with pixel guards is NP-hard in the general and all-or-nothing models, even for simple polyominoes.

For the special cases of thin polyomino paths and trees in the all-or-nothing visibility model, we give optimal algorithms for guarding paths in linear time and an optimal algorithm for guarding trees with unlimited visibility and with limited pixel guard visibility. However, we conjecture that guarding thin polyomino trees has a polynomial-time (exact) algorithm under the general model.

It remains open whether there is any approximation algorithm for guarding simple polyominoes.

Chapter 4

Guarding Polyforms[1]

4.1 Introduction

A *polyform* is a plane figure or solid compound comprised of a collection of m edge- or face-aligned copies (pixels or voxels) of a given shape. Examples include *polyominoes*, *polyiamonds*, and *polyhexes*, which have base shapes of squares, equilateral triangles and regular hexagons, respectively. We may use the notation (m) to indicate an m-polyform (a polyform with m cells). Biedl et al. study the combinatorics of guarding m-polyominoes [8]. Both *point guards* and guards that are themselves entire pixels of the polyomino (*pixel guards*), are considered in their work. A polyomino (or any polyform) P is considered *covered* if every point on its interior is seen by at least one guard g of a guard set G, i.e., $\forall p \in P, \exists g \in G$ such that $\overline{gp} \subset P$. If pixel guards are used, then $\overline{gp} \subset P$ means there is a point of g such that the line segment defined by this point and p is in P. Also, different types of visibility are explored including the standard unrestricted model and the more restrictive *all-or-nothing* model. In the latter case, a pixel p is guarded only if a single point guard a (or point a inside a pixel guard) sees all points of p (i.e., $ab \subset P$, for all $b \in P$). The authors demonstrate that $\lfloor \frac{m+1}{3} \rfloor$ point guards and $\left\lfloor \frac{3m}{11} \right\rfloor + 1$ pixel guards are always sufficient and sometimes necessary to cover an m-polyomino.

The 3D analog of a polyomino, a *polycube*, is a connected union of face-aligned unit cubes. An open question posed by Biedl et al. is "What are necessary and sufficient conditions for guarding polycubes?" [8] In this study,

[1] A preliminary version appears in *1st Computational Geometry: Young Researchers Forum 2012* [30].

Polyform	Guard type	Lower bound	Upper bound
polycube	point	$\lfloor \frac{m+1}{3} \rfloor$	$\lfloor \frac{m+1}{3} \rfloor$
	voxel	$\approx \frac{5m}{17}$	$\lfloor \frac{4m}{13} \rfloor + 1$
polyiamond	point	$\approx \frac{2m}{11}$	$\lfloor \frac{2m}{9} \rfloor + 1$
	pixel	$\approx \frac{m}{7}$	$\lfloor \frac{3m}{16} \rfloor + 1$
polyhex	point	$\lfloor \frac{m}{2} \rfloor$	$\lfloor \frac{m}{2} \rfloor$
	pixel	$\approx \frac{5m}{14}$	$\lfloor \frac{3m}{7} \rfloor + 1$

Table 4.1: Combinatorial bounds for guarding polycubes, polyiamonds, and polyhexes

we show that $\lfloor \frac{m+1}{3} \rfloor$ point guards are sometimes necessary and always sufficient to cover an m-polycube, matching the bounds for guarding an m-polyomino. Additionally, we provide combinatorial bounds for guarding polycubes with *voxel guards* and polyiamonds and polyhexes with both point and pixel guards. We summarize our results in Table 4.1. In Section 4.2 we give constructions that achieve the reported lower bounds. In Section 4.3 we establish upper bounds. We conclude with a discussion and some open problems in Section 4.4.

4.2 Necessary Conditions

For each of the polyforms in consideration (polycubes, polyiamonds and polyhexes), we establish constructions that achieve the reported number of guards that are sometimes necessary to cover the polyform. We begin with polycubes.

4.2.1 Polycubes

We see from Figure 4.1 (top) that $\lfloor \frac{m+1}{3} \rfloor$ point guards are sometimes necessary to cover a polycube on m voxels. In the voxel guard case, we obtain a lower bound of $\approx \frac{5m}{17}$.[2] We can acquire the $\frac{5m}{17}$ bound by linking spiral-shaped 13-

[2]To be precise, we can use this pattern to construct polycubes that require $f(m) = \lfloor \frac{m+1}{17} \rfloor + \lfloor \frac{m+4}{17} \rfloor + \lfloor \frac{m+7}{17} \rfloor + \lfloor \frac{m+11}{17} \rfloor + \lfloor \frac{m+15}{17} \rfloor \approx \frac{5m}{17}$ voxel guards. However, this is not of great significance since there is a gap between our lower and upper bounds and $5m/17$ and $f(m)$ never differ by more than 1. Therefore, for this and the other cases in which we have

polycubes via T-shaped 4-polycubes as depicted at the bottom of Figure 4.1. The union of a spiral-shaped polycube and a T-shaped polycube gives us a (17) that requires 5 voxel guards.

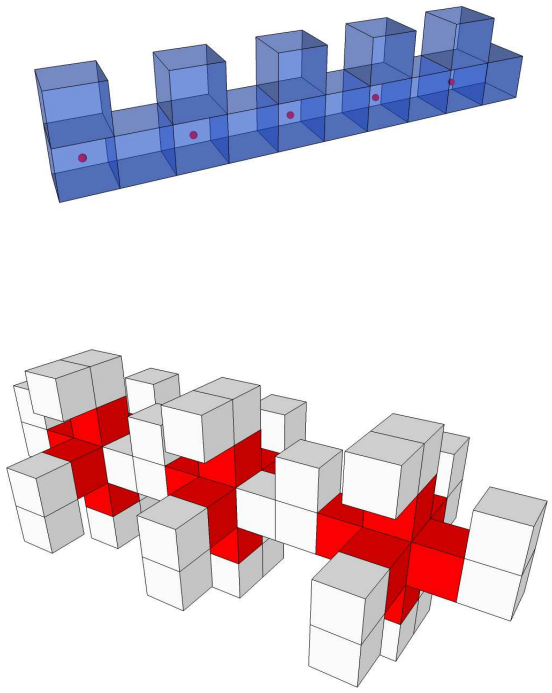


Figure 4.1: Lower bound constructions for polycubes

4.2.2 Polyiamonds

We see from Figure 4.2 (left) that $\approx \frac{2m}{11}$ point guards are sometimes necessary to cover a polyiamond. Figure 4.2 (right) demonstrates that $\approx \frac{m}{7}$ pixel guards are sometimes necessary to cover a polyiamond.

gaps, we will not trouble ourselves with the exact functions illustrated by our lower bound constructions.

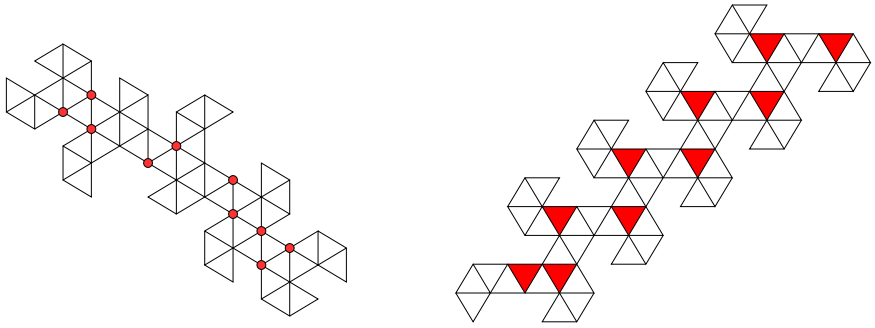


Figure 4.2: Lower bound constructions for polyiamonds

4.2.3 Polyhexes

We observe from Figure 4.3 that $\lfloor \frac{m}{2} \rfloor$ point guards (top) and $\approx \frac{5m}{14}$ pixel guards (bottom) are sometimes necessary to cover an m-polyhex. The 14-polyhex outlined on the bottom requires 5 pixel guards. Repeated copies of this subpolyhex can give us larger lower bound constructions.

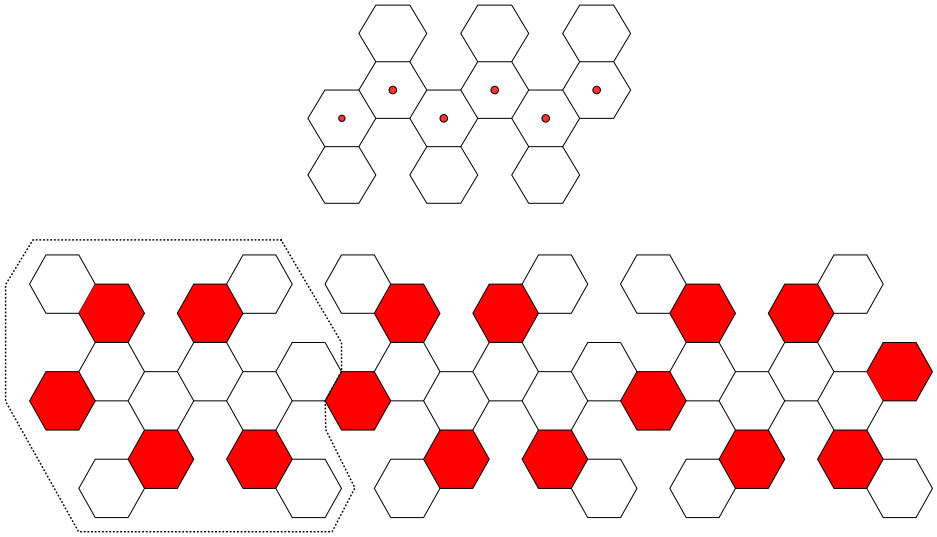


Figure 4.3: Lower bound constructions for polyhexes

4.3 Sufficiency Conditions

Many of the sufficiency conditions for the various polyforms are obtained by arguing that we can create a 'smart' BFS tree and iteratively remove certain types of subpolyforms while leaving the remaining polyform connected (via extensive case analysis) as in [8]. We will first study the upper bounds for guarding polycubes.

4.3.1 Polycubes

Point Guards

Lemma 4.3.1. *Every polycube P with 5 voxels is star-shaped except, $5'$, $5''$ and $5'''$.*

Proof. Let r be a voxel of P with maximum degree. If r has 4 neighbors, a point placed at any corner of r will cover all of P. If r has 3 neighbors, let p be the voxel adjacent to r that is adjacent to another voxel q. By placing a point guard at a corner of r shared by p, all of P will be covered. Finally, suppose r has two neighbors. In this case P is either a rectangular prism, in which case a point guard placed at any corner of P will do, or there is exactly one, two, or three sets of three voxels in P that each form a turn. In the one turn case, we can simply place a guard at a corner shared by these three voxels and P is covered. If there are two turns and the middle voxels in these turns are adjacent, then placing a point guard at a common vertex shared by these voxels suffices. If the two middle voxels in the two turns are not adjacent, then P would be $5'$, $5''$ or $5'''$: a contradiction. In the three turn case, we place a point guard at a vertex shared by the center voxel of the path and its two neighbors. □

Theorem 4.3.2. *For $m \geq 2$, $\lfloor \frac{m+1}{3} \rfloor$ point guards are always sufficient to cover a polycube P with m voxels and h holes, even in the all-or-nothing model.*

Proof. We compute a BFS-tree constructed from P's dual graph rooted at a vertex of degree ≤ 3 using a special rule analogous to the polyomino version (turns are given preference; see Chapter 3). Let T be the BFS tree, and for any node v in T, let T_v be the subtree rooted at v (v and all its descendants). We will show that there is some rooted subtree T_v having v vertices that can

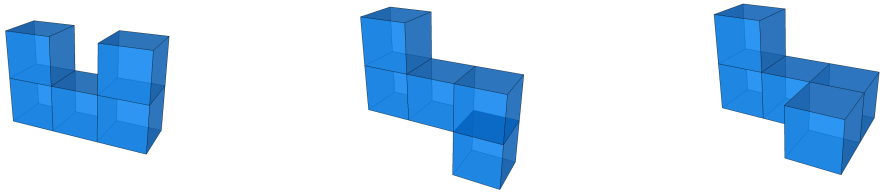


Figure 4.4: The three (5)'s that require 2 point guards: 5′, 5″, and 5‴, from left to right

be covered with $\lfloor \frac{m+1}{3} \rfloor$ point guards. We call such a subtree *voxel-good*. Note that all vertices $v \in T$ can have at most 5 children since the corresponding voxel of v has 6 faces.

Let q be a lowest leaf of T, let p_1 be its parent, p_2 be its grandparent, p_3 be its great-grandparent, and generally p_i be its great-great-...-great-grandparent, where "great" is repeated $i-2$ times. Through an extensive case analysis, we will show that some rooted subtree of T_{p_5} is voxel-good. We begin with T_{p_1} and work our way up generations as needed.

T_{p_1} **(and other subtrees of height** ≤ 1**):** If $|T_{p_1}| \geq 3$, then we can cover everything by placing a point guard at any corner of p_1: The union of p_1 and any child is a convex rectangular prism, and hence star-shaped. Hence, T_{p_1} is voxel-good. Else, $|T_{p_1}| = 2$ and we go up a generation.

T_{p_2} **(and other subtrees of height** ≤ 2**):** If any subtree of the children of T_{p_2} is voxel-good, we remove it and continue. Else, we consider several cases:

(2a) p_1 has no siblings. Then $|T_{p_2}| = 3$ and we place a point guard at any corner of p_1 so that T_{p_2} is voxel-good.

(2b) p_1 has at least one sibling and all of p_1's siblings are leaves. By placing a point guard at a corner shared by p_1 and p_2, all voxels will be covered. Thus, we use 1 point guard for between 4 and 6 voxels and T_{p_2} is voxel-good.

(2c) p_1 has at least one sibling and at least one of these siblings has a child itself. Then $5 \leq |T_{p_2}| \leq 11$. If p_1 has one sibling so that $|T_{p_2}| = 5$, then

T_{p_2} is star-shaped unless it is $5'$, $5''$, or $5'''$ by Lemma 4.3.1. If T_{p_2} is star-shaped in this case, then it is voxel-good. Else it is one of $5'$, $5''$, or $5'''$ and we go up a generation. Otherwise, $|T_{p_2}| \geq 6$ and two guards suffice: place two guards at a pair of non-adjacent corners of p_2 shared by p_3 (p_2's parent). Then we have also placed a point guard in every child of p_2. Since the union of every child of p_2 and that child's (possible) only child is a rectangular convex prism containing a point guard, T_{p_2} is covered and hence voxel-good.

T_{p_3} (and other subtrees of height ≤ 3): If any subtree of the children of T_{p_3} is voxel-good, we remove it and continue. Else, we consider several cases:

(3a) p_2 has no siblings. Then we place two point guards at non-adjacent corners of p_2 shared by p_3 so that T_{p_3} is covered and hence voxel-good.

(3b) p_2 has one sibling, p_2' and p_2' is a leaf. Then by placing a pair of point guards at non-adjacent corners of p_2 shared by p_3, all of T_{p_3} is covered and hence voxel-good.

(3c) p_2 has one sibling, p_2' and p_2' has exactly one child, p_1''. If p_2-p_3-p_2' forms a turn or if p_2-p_3-p_2'-p_1'' is straight, then we can place two point guards at the corners shared by p_2 and p_3 and T_{p_3} is voxel-good. If p_2-p_3-p_2' is straight and p_3-p_2'-p_1'' forms a turn (see Figure 4.5), then we go up a generation.

(3d) p_2 has one sibling, p_2' and p_2' is a $5'$, $5''$, or $5'''$. If p_2-p_3-p_2' forms a turn, then we place two point guards at the vertices of the polycube edge shared by p_2 and p_2'. We then find that T_{p_3} is covered and hence voxel-good. Otherwise, p_2-p_3-p_2' makes a straight path and we go up a generation (see Figure 4.6).

(3e) p_2 has two siblings, p_2' and p_2'', which are both leaves. Then we place two guards at non-adjacent corners shared by p_2 and p_3 and T_{p_3} is voxel-good.

(3f) p_2 has two siblings, p_2' and p_2'' and one or both have exactly one child. Then we have a (9) or (10) that is coverable with at most 3 point guards: place two point guards at non-adjacent corners shared by p_2 and p_3 so that T_{p_2} is covered and one of the trees rooted at p_2' or p_2'' is covered (whichever forms a turn with p_2 and p_3). We then place a third point guard if necessary at a corner of whichever voxel, p_2' or p_2'', is the root of the possibly remaining uncovered subtree. T_{p_3} is then voxel-good.

(3g) p_2 has two siblings, one of which, say p_2'' without loss of generality, is a $5'$, $5''$, or $5'''$ and the other, p_2' is a leaf or has exactly one child. Then we have a polycube with 12 (or 13) voxels and by placing 4 point guards at 4 mutually non-adjacent corners of p_3, we cover all voxels and T_{p_3} is voxel-good.

(3h) p_2 has two siblings, both of which are a $5'$, $5''$, or $5'''$. We can cover T_{p_3} using the same guard placement as in the previous case. Hence, T_{p_3} is voxel-good in this case as well.

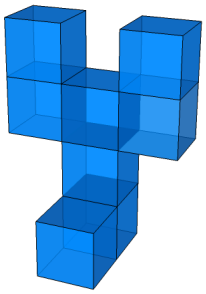


Figure 4.5: An example of an (8) that requires 3 point guards and is hence not voxel-good

T_{p_4} **(and other subtrees of height** ≤ 4**):** If any subtree of the children of T_{p_4} is voxel-good, we remove it and continue. Else, we consider several cases. The first set of cases will involve situations where T_{p_3} is an (8) requiring 3 point guards (see case (3c) above).

(4a) p_3 has no siblings. Then three guards suffice to cover T_{p_4}, which has 9 voxels and so T_{p_4} is voxel-good.

(4b) p_3 has exactly one sibling, p_3' and p_3' is a leaf. Then we place two guards at the two corners shared by p_3, the child of p_3 that is a $5'$, $5''$, or $5'''$ and p_4 along with a third guard at a corner shared by p_3 and the child of p_3 that is a (2). Clearly, T_{p_4} is covered and since we use 3 guards for $|T_{p_4}| = 10$ voxels, T_{p_4} is voxel-good.

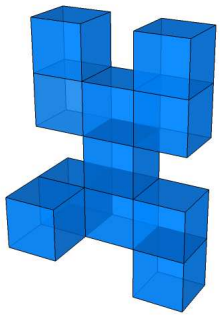


Figure 4.6: An example of an (11) that requires 4 point guards and is hence not voxel-good

(4c) p_3 has exactly one sibling, p_3', and p_3' has exactly one child. Then this child is either covered using the previous placement of three guards or p_4-p_3'-p_3''s child makes a turn and this child is not visible, which would require us to place another guard. However, in this case, we have an (11) requiring 4 point guards and we are forced to go up a generation (see left image in Figure 4.7).

(4d) p_3 has exactly one sibling, p_3' and T'_{p_3} is a $5'$, $5''$, or $5'''$. Then we have a (14) requiring 5 point guards and so we go up a generation (see middle image in Figure 4.7).

(4e) p_3 has exactly one sibling, p_3' and T'_{p_3} is an (8) requiring three point guards. Then we have a (17) requiring 6 point guards and so again we go up a generation (see right image in Figure 4.7).

(4f) p_3 has two siblings, p_3' and p_3'', which are both leaves. Then we place two point guards at the corners shared by p_4, p_3, and the child of p_3 that is a $5'$, $5''$ or $5'''$, and then at one more additional guard at another corner shared by p_4 and p_3. Since T_{p_4} has 11 voxels and requires 3 guards, it is voxel-good.

(4g) p_3 has two siblings, p_3' and p_3'', one of which is a leaf (or has a single child) while the other is not a leaf. Then the only seemingly 'problematic'

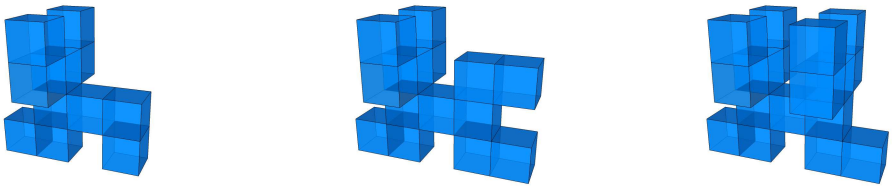


Figure 4.7: Examples of cases when we need to go up a level from p_4. Left: 11 voxels and 4 guards required, Middle: 14 voxels and 5 guards required, and Right: 17 voxels and 6 guards required.

situations would be the ones similar to those illustrated in Figure 4.7, except with an additional voxel adjacent to p_4. However, T_{p_4} is voxel-good in each case because the sibling that is a leaf is trivially covered and so we have guard to voxel ratios of 12:4, 15:5, and 18:6, respectively. It is easy to see that T_{p_4} is voxel-good even when the sibling that is a leaf actually has a single child appended to it. Since not both of p_3' and p_3'' can be the root of a 5′, 5″ or 5‴ or an (8) requiring 3 point guards, some subtree of T_{p_4} must be voxel-good.

We now move to the case where T_{p_3} is an (11) requiring 4 point guards to cover from case (3d) (see Figure 4.6).

(4h) p_3 has no siblings. Then 4 guards suffice by placing them at the corners shared by p_3 and p_4. T_{p_4} is then voxel-good.

(4i) p_3 has exactly one sibling, p_3' and p_3' is a leaf. Then placing 4 guards as before will cover T_{p_4} and so T_{p_4} is voxel-good.

(4j) p_3 has exactly one sibling, p_3', which has one of the following configurations: has a child that is not along a straight path through p_4 and p_3', is the root of a 5, 5″, or 5‴, or is an (8) requiring 3 point guards, or is an (11) requiring 4 point guards (as p_3 does). In any of these four subcases, we go up a generation (see Figure 4.8).

(4k) p_3 has two siblings, p_3' and p_3'', which are roots of subtrees of size at most 2. Then we place 4 guards at the corners shared by p_3 and p_4 and T_{p_4} is voxel-good.

(4l) p_3 has two siblings, p_3' and p_3'' and p_3' is a 5, 5″, or 5‴ while p_3'' is a leaf or has exactly one child. Then $|T_{p_4}| = 18$ or 19 and is coverable with 6 guards: place 4 at the corners shared by by p_3 and p_4 and then two more at corners of p_4 so that the 5, 5″, or 5‴ rooted at p_3' is covered. T_{p_4} is then voxel-good.

(4m) p_3 has two siblings, p_3' and p_3'', and p_3' is rooted at an (11) requiring 4 point guards. In this case $|T_{p_4}| = 24$ or 25 and 8 guards suffice: one at each corner of p_4. Therefore, T_{p_4} is voxel-good.

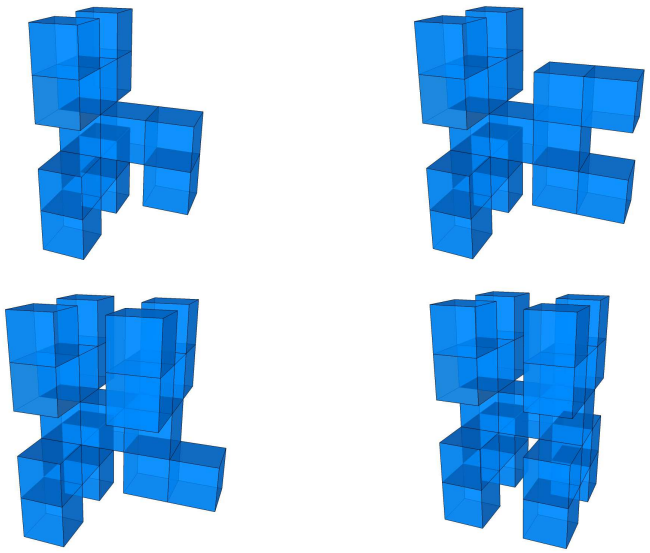


Figure 4.8: More examples of cases when we need to go up a level from p_4. Top left: 14 voxels and 5 guards required, Top right: 17 voxels and 6 guards required, Bottom left: 20 voxels and 7 guards required, and Bottom right: 23 voxels and 8 guards required.

T_{p_5} **(and other subtrees of height** ≤ 5**):** If any subtree of the children of T_{p_5} is voxel-good, we remove it and continue. Else, we consider several cases. We first consider the cases (4c), (4d), and (4e), where T_{p_3} is an (8).

(5a) p_4 has no siblings. Then we have one of the following ratios, all of which show that T_{p_5} is voxel-good: 4:12, 5:15, or 6:18.

(5b) p_4 has exactly one sibling, p'_4 (it can't have two or more siblings due to the geometry of T_{p_4}), which is a leaf, and $p_4 - p_5 - p'_4$ makes a turn. Then p'_4 is trivially covered with the 4, 5, or 6 guards used for the three possible cases ((4c), (4d), or (4e), respectively) and T_{p_5} is voxel-good.

(5c) p_4 has a sibling, p'_4, which has exactly one child (it can't have more than one child again due to the geometry) and $p_4 - p_5 - p'_4$ makes a turn. In this case we reach a contradiction: $T_{p'''_3}$ isn't voxel-good, but p'''_3 has no siblings so appending one voxel (p'_4) will make for a voxel-good subpolycube as shown previously in our analysis for T_{p_4}.

(5d) p_4 has a sibling, p'_4, which is a leaf, and $p_4 - p_5 - p'_4$ is straight. This can be guarded with the 4, 5, or 6 guards used for guarding cases (4c), (4d) or (4e) and T_{p_5} is voxel-good.

(5e) p_4 has a sibling, p'_4, which has exactly one child. We make the observation that p'_4 and p_1 or p'_1 must share a corner. Placing a guard at this corner, say p_1, along with one at the corner shared by $p'_1$and p_4, one at a corner shared by p''_2 and p_4, and at a corner shared by p'''_2 and p_4 covers T_{p_5} with a 4:14, 5:17, or 6:20 ratio, respectively so T_{p_5} is voxel-good.

(5f) p_4 has a sibling, p'_4 and $T_{p'_4}$ is a $5'$, $5''$, or $5'''$. Then we use the same guard placement as before and place one additional guard in $T_{p'_4}$ to cover its other 'prong'. This leads to a 5:17, 6:20, or 7:23 guard to voxel ratio and so T_{p_5} is voxel-good.

(5g) p_4 has a sibling, p'_4 and $T_{p'_4}$ is an (8) requiring 3 guards to cover. Then again we only need two new guards since a guard at the corner shared by p_1 or p'_1 and p'_4 will cover the everything except the (5) subtree that needs 2 guards. This gives a 6:20, 7:23, or 7:26 guard to voxel ratio and so T_{p_5} is voxel-good.

(5h) p_4 has a sibling, p'_4 and $T_{p'_4}$ is an (11) requiring 4 guards to cover, then this can only be realized if T_{p_4} is also an (11) (as in case (4c)). This polycube rooted at T_{p_5} requires 6 guards (see Figure 4.9) and is voxel-good.

The two other possible cases of interest would be when $T_{p'_4}$ is a (14) requiring 5 guards to cover or a (17) requiring 6 guards. However, neither of these can be realized as a polycube.

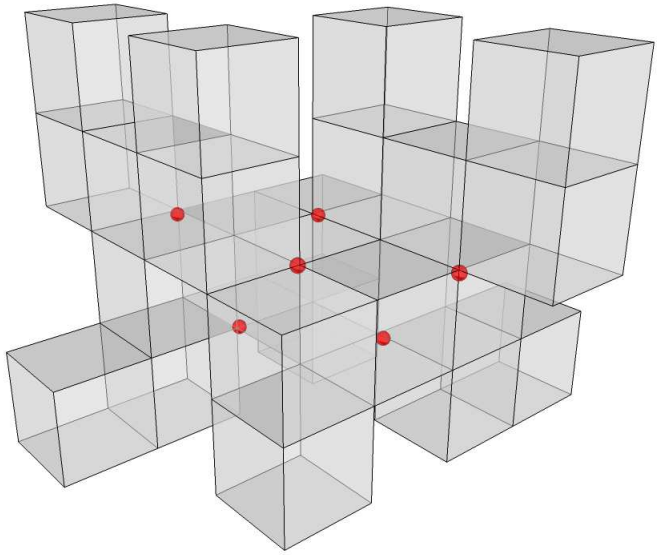


Figure 4.9: When T_{p_4} and $T_{p_4'}$ are both (11)'s that individually require 4 guards, linking them by a single voxel, p_5, creating a (23) requires just 6 point guards as depicted.

Finally, we consider the four unresolved subcases when T_{p_3} is an (11) from (4j). We observe that p_4 can have no siblings due to the geometry of the polycube. Since p_5 is covered by placing all guards at appropriate corners of p_4, we have a 5:15, 6:18 or 7:21 guard to voxel ratio and so T_{p_5} is voxel-good.

We've seen that we can always recursively find a subtree that is voxel-good. Hence we have at worst a 1:3 ratio between point guards and voxels. Elementary calculations analogous to those done for the polyomino version in Chapter 3 show that this implies that $\lfloor \frac{m+1}{3} \rfloor$ point guards always suffice. Since the guards never needed to 'team-up' to cover a voxel, our bound holds in the all-or-nothing model as well. □

Corollary 4.3.3. *For a polycube P with $m \geq 2$ voxels and h holes, $\lfloor \frac{m+1}{3} \rfloor$ point guards are always sufficient and sometimes necessary to cover P, even in the all-or-nothing model.*

Voxel Guards

We now investigate the voxel guard case. Our approach will be analogous: use a BFS tree rooted at some vertex of degree 3 or more and iteratively remove subtrees that are 'voxel-good'. First we consider two lemmas that will be useful. The three special polycubes with six voxels that require two voxel guards can be found in Figure 4.10.

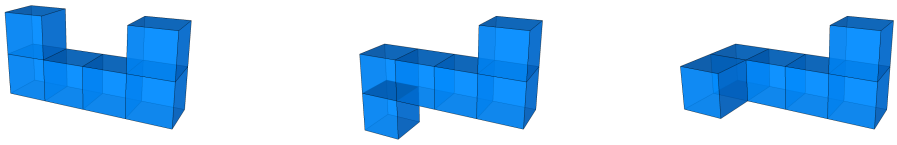


Figure 4.10: The three (6)'s that require 2 voxel guards: $6'$, $6''$, and $6'''$, from left to right. The two 'turn' voxels are separated by two 'straight' voxels in each case.

Lemma 4.3.4. *Every polycube P with 6 voxels is star-shaped except, $6'$, $6''$ and $6'''$.*

Proof. Let r be a voxel of P with maximum degree. If r has 3, 4 or 5 neighbors, place a guard at r and clearly all of P will be covered. If r has two neighbors we consider a few subcases.

First, if P has no turn voxels (a voxel with neighbors at two adjacent faces), then it is a rectangular prism and hence star-shaped: a guard at any voxel will do.

If P has exactly one turn voxel, then we place a guard in the turn voxel and P is covered.

If P has exactly two turn voxels, we consider a few subcases: If these two turn voxels are adjacent, then placing a guard in either of the turn voxels will cover P since the guard will have points in both turn voxels and hence cover all straight voxel paths extending from these turn voxels. If these two turn voxels are separated by one straight voxel, then placing a guard at this separating straight voxel will cover P. If these two turn voxels are separated by two straight voxels, then we must have a $6'$, $6''$ or $6'''$. It is not possible to

separate two turn voxels by three or more straight voxels in a voxel path of length 6.

Suppose P has exactly three turn voxels. Then in the subcase where two of these turn voxels are adjacent and the third is separated by a straight voxel, we place a guard in this separating straight voxel and P is covered since the guard will have at least one point in all three turn voxels. Else, the three turn voxels form a subpath of length three. By placing a guard at the middle voxel of this subpath, all of P will again be covered for the same reason.

If P has (the maximum) four turn voxels, then placing a guard in either of the 'middle' two voxels will cover P since the guard will have at least one point in all four turn voxels. □

Lemma 4.3.5. *A subpolycube whose dual graph has a diameter ≤ 5 can be covered by single voxel guard.*

Proof. Let Q be a subpolycube of a polycube P with diameter ≤ 5. Find a path γ in Q that achieves this diameter and place a voxel guard g at the middle voxel (if the diameter is even, place the guard at either of the centermost voxels). Clearly, g sees this center voxel and any adjacent voxels. Since points of g are also in all of these adjacent voxels, any neighbors of these voxels are also covered. □

Theorem 4.3.6. *For $m \geq 1$, $\lfloor \frac{4m}{13} \rfloor + 1$ voxel guards are always sufficient to cover a polycube P with m voxels and h holes, even in the all-or-nothing model.*

Proof. Again, we compute a BFS-tree constructed from P's dual graph rooted at a vertex of degree ≤ 3. Through case analysis, we will show that some rooted subtree of T_{p_4} is voxel-good (has at worst a 4:13 guard to voxel ratio). We begin with T_{p_1} and work our way up generations as needed.

T_{p_1} (and other subtrees of height ≤ 1): If $|T_{p_1}| \geq 4$, then a guard placed at p_1 will cover p_1 and all its children. Hence, T_{p_1} is voxel-good. Else, we go up a generation.

T_{p_2} (and other subtrees of height ≤ 2): If $|T_{p_2}| \geq 4$, then we can place a guard at p_2 so that points of this guard will be in the children of p_2. Since the union of one of these children and any of this child's children is a rectangular

prism, P is covered and T_{p_2} is voxel-good. Else, we go up a generation.

T_{p_3} (and other subtrees of height ≤ 3): If any subtree of the children of T_{p_3} is voxel-good, we remove it and continue. Else, we consider several cases:

(3a) p_2 has no siblings. Then $|T_{p_3}| = 4$ and we place a guard at p_2 and see that T_{p_3} is voxel-good.

(3b) p_2 has at least one sibling and all of p_2's siblings are leaves. Then we can cover P by placing a guard at p_2. T_{p_3} is voxel-good.

(3c) p_2 has exactly one sibling and T_{p_3} is a path of length 6. Then T_{p_3} is star-shaped unless it is a $6'$, $6''$ or $6'''$, in which case we go up a generation.

(3d) p_2 has exactly one sibling and $|T_{p_3}| = 7$. Then two guards suffice placed at p_2 and p_2' and T_{p_3} is voxel-good.

(3e) p_2 has exactly two siblings, p_2' and p_2'', and one sibling, say p_2' has a single child. Then we have a (7) that is coverable with two guards: place one at p_2 and the other at p_2'. Hence, T_{p_3} is voxel-good.

(3f) p_2 has exactly two siblings, one of which is a leaf while the other, say p_2', is the root of a subtree of size 3. Then again, placing guards at p_2 and p_2' covers all of T_{p_3} so that T_{p_3} is voxel-good.

(3g) p_2 has exactly two siblings, both of which have a single child. Then two guards suffice by placing them at p_2 and p_3 so that T_{p_3} is voxel-good.

(3h) p_2 has exactly two siblings, p_2' is the root of a subtree of size 3 while p_2'' is the root of a subtree of of size 2. Two guards suffice by placing them at p_2 and p_2' and T_{p_3} is voxel-good (see Figure 4.11 for an illustration).

(3i) p_2 has exactly two siblings, both of which are roots of subtrees of size 3 that are both not paths. Two guards suffice: place them at p_2 and p_3 and T_{p_3} is voxel-good.

(3j) p_2 has exactly two siblings, both of which are roots of subtrees of size 3 with one subtree, say the one rooted at p_2', being a path and the other not a path. We may place two guards at p_2 and p_2' and T_{p_3} is voxel-good.

(3l) p_2 has exactly two siblings, both of which are roots of subtrees of size 3 that are paths. In this case, we may need 3 guards, giving us a 3:10 guard to voxel ratio. However, since $3/10 < 4/13$, T_{p_3} is voxel good.

(3m) p_2 has exactly three siblings and each sibling has exactly one or two children that are leaves. Here, one guard at p_2 and p_3 will suffice and T_{p_3} is voxel-good.

(3n) p_2 has exactly three siblings, one of these siblings is the root of a path of length 2, and the other two siblings have exactly one or two children that are leaves. Then 3 guards suffice: one at p_2, one at p_3 and a third at p_2', the root of the path of length 3. This gives a 3:13 guard to voxel ratio and so T_{p_3} is voxel-good.

(3o) p_2 has exactly three siblings, two of which are roots of paths of length 3 and the third has exactly one or two children that are leaves. In this case, 3 guards suffice placed at p_2 and at the siblings that are roots of paths of length 3. Hence T_{p_3} is voxel-good.

(3p) p_2 has exactly three siblings, all of which are roots of paths of length 3. In this case at most 4 guards may be necessary to cover the 13 voxels, so T_{p_3} is voxel-good.

(3q) p_2 has exactly four siblings, all of which are roots of subtrees with one or two children that are leaves. Two guards suffice by placing them at p_2 and p_3 and T_{p_3} is voxel-good.

(3r) p_2 has exactly four siblings, all of which are roots of subtrees that are paths of length 3. Then we require 5 guards to cover the 16 voxels. Since $5/16 > 4/13$, we go up a generation. We observe however that p_3 can have no siblings due to our BFS convention. Hence we can achieve a 5:17 guard to voxel ratio and T_{p_4} is voxel-good.

We see that one can always recursively remove a subpolycube that is voxel-good (i.e., no ratio worse than 4:13 is required). For the base case, we may need an additional voxel guard if we have between 1 and 3 voxels left over. Hence we have that $\lfloor \frac{4m}{13} \rfloor + 1$ voxel guards always suffice. Since the guards never needed to 'team-up' to cover a voxel, our bound holds in the all-or-nothing model as well. □

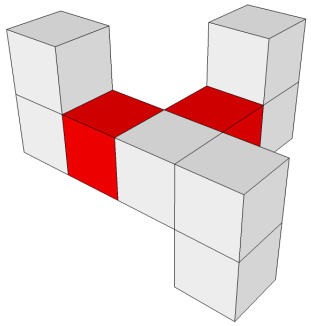


Figure 4.11: An example situation described by case (3h) in which guards placed at p_2 and p_2' cover the polycube. p_3 is the voxel with degree 3.

Corollary 4.3.7. *For a polycube P with $m \geq 4$ voxels and h holes, $\approx \frac{5m}{17}$ voxel guards are sometimes necessary while $\lfloor \frac{4m}{13} \rfloor + 1$ voxel guards are always sufficient to cover P, even in the all-or-nothing model.*

Observation 1. *As in the polyomino case using pixel guards, open voxel guards are significantly weaker. Nearly identical arguments can be made to show that $\lfloor \frac{m}{2} \rfloor$ open voxel guards are sometimes necessary and always sufficient to cover an m-polycube.*

Observation 2. *The combinatorial bounds for polycubes extend to the case when its voxels are more general, face-aligned rectangular prisms. The arguments are entirely analgous to the extension of the combinatorial bounds of polyominoes to rectanglominoes (see Section 3.2.3).*

4.3.2 Polyiamonds

In each of the two following proofs of sufficiency, we construct a BFS tree rooted at a some vertex of degree ≤ 2. No special preference to the order in which children of a pixel are explored is required. Note that in the BFS tree, a vertex v can have at most 2 children since the pixels have three sides. Our approach in each case will be to iteratively remove subtrees that are 'pixel-good' (except perhaps, during the final iteration), which in this case means that we have at worst a 2:9 ratio and 3:16 ratio between guards used and pixels covered in the point and pixel guard cases, respectively.

Point Guards

We begin with a couple of useful lemmas:

Lemma 4.3.8. *Let T be a BFS tree of the dual of a polyiamond P having a root node with degree ≤ 2 and height ≤ 3. Then T can be covered with a single point guard at the vertex of r shared by its (possibly two) children.*

Proof. Consider any leaf node q in T. The unique subpolyiamond S formed by the path in T from q up to but not including r has between 0 and 3 pixels. Therefore, S is either empty, a single equilateral triangle, a rhombus, or a trapezoid; all of which are convex. Since the point guard placed in r is also a vertex of S, S is covered. □

Lemma 4.3.9. *Let P be an 8-polyiamond such that the dual of P is a path. Then P can be covered with a single point guard unless P is $8'$, $8''$ or $8'''$ (see Figure 4.12).*

Proof. All 15 free 8-polyiamonds with duals that are paths are shown in Figure 4.12. By inspection we see that all require a single point guard except $8'$, $8''$ and $8'''$, which each require 2 point guards. □

Theorem 4.3.10. *For $m \geq 1$, $\lfloor \frac{2m}{9} \rfloor + 1$ point guards are always sufficient to cover a polyiamond P with m pixels and h holes, even in the all-or-nothing model.*

Proof. Compute a BFS tree T constructed from P's dual graph rooted at a vertex of degree ≤ 2. Let q be a lowest leaf of T, let p_1 be its parent, p_2 be its grandparent, p_3 be its great-grandparent, and generally p_i be its great-great-...-great-grandparent, where "great" is repeated $i-2$ times. Through case analysis, we will show that some rooted subtree of T_{p_6} is pixel-good. We begin with T_{p_2} and work our way up generations as needed.

T_{p_2} (and other subtrees of height ≤ 2): If $5 \leq |T_{p_2}| \leq 7$, then we place one guard at the point of p_2 shared by its (possibly two) children and by Lemma 4.3.8, T_{p_2} is covered. At worst, this gives a 1:5 ratio and so T_{p_2} is pixel-good. Else, $|T_{p_2}| = 3$ or 4 and we go up a generation.

T_{p_3} (and other subtrees of height ≤ 3): If any subtree of the children of T_{p_3} is pixel-good, we remove it and continue. Else, we consider several cases:

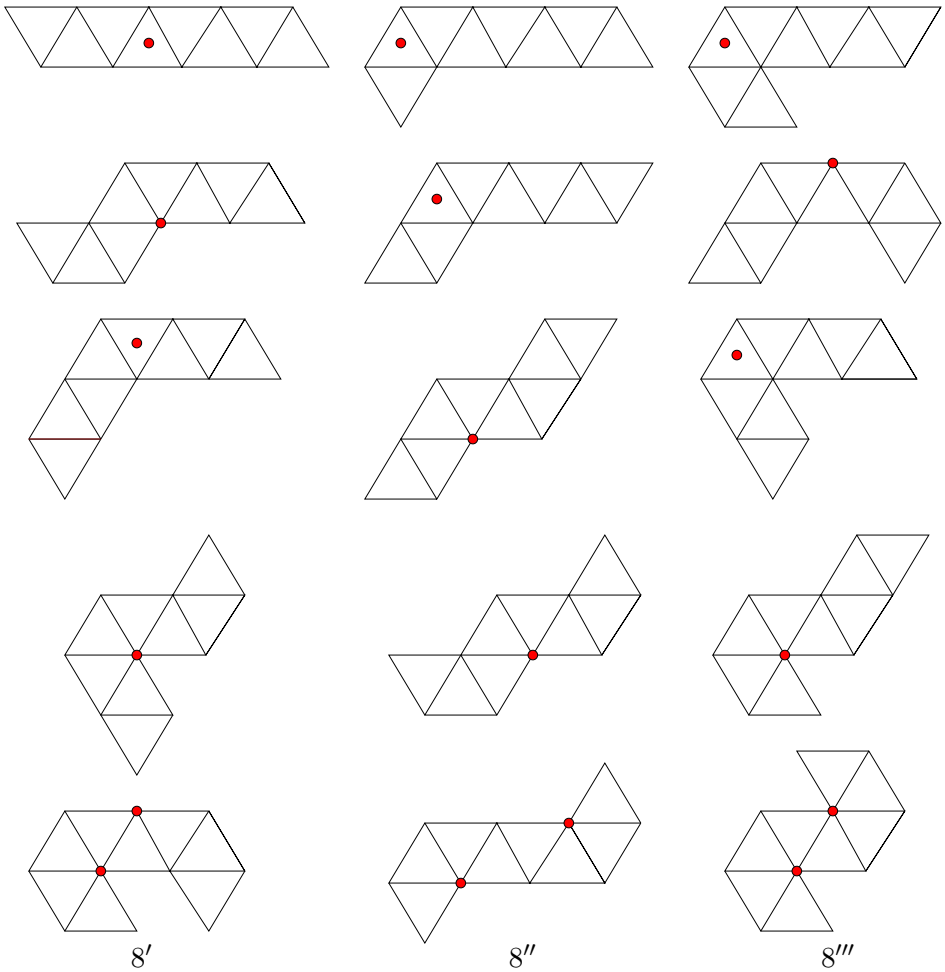


Figure 4.12: The 15 free 8-Polyiamonds with duals that are paths. Only $8'$, $8''$ and $8'''$ require 2 point guards. The shaded dots represent possible guard placements that cover the polyiamonds.

(3a) p_3 has just a single child, p_2. Then $|T_{p_3}| = 4$ and we go up a generation.

(3b) p_3 has two children. Then since T_{p_3} has height = 3, it can be covered with a single point guard by Lemma 4.3.8. At worst, this gives a 1:5 ratio and so T_{p_2} is pixel-good.

T_{p_4} **(and other subtrees of height** ≤ 4**):** If any subtree of the children of T_{p_4} is pixel-good, we remove it and continue. Else, we consider several cases:

(4a) p_4 has just a single child, p_3. Then T_{p_4} is a path of length 5. If we make p_2 the root of a new BFS tree T' of T_{p_4}, then by Lemma 4.3.8 T_{p_4} requires just a single guard yielding a 1:5 ratio and so T_{p_4} is pixel-good.

(4b) p_4 has two children p_3 and p'_3 with $T_{p'_3}$ having a height of 0 or 1. (In this case, we create a new BFS tree T' rooted at p_3). By Lemma 4.3.8 T_{p_4} requires just a single guard yielding a 1:6 ratio at worst and so T_{p_4} is pixel-good.

(4c) p_4 has two children p_3 and p'_3 with $T_{p'_3}$ being a path of length 3. In this case, $|T_{p_4}| = 8$. If T_{p_4} is not $8'$, $8''$ or $8'''$, then by Lemma 4.3.9, T_{p_4} can be covered with a single guard and is hence pixel-good. Else, T_{p_4} is $8'$, $8''$ or $8'''$ and we go up a generation.

(4d) p_4 has two children p_3 and p'_3 with $|T_{p'_3}| = 4$. Then T_{p_4} we place one point guard at common vertex of p_3 and p_2 and another at a common vertex of p'_3 and a child p_3. This gives us a 2:9 ratio and T_{p_4} is pixel-good.

T_{p_5} **(and other subtrees of height** ≤ 5**):** If any subtree of the children of T_{p_5} is pixel-good, we remove it and continue. Else, we consider several cases:

(5a) T_{p_4} is an $8'''$. Then p_5 cannot have a parent without violating the order in which pixels were discovered by the BFS tree. We use two point guards on this final 8-polyiamond.

(5b) T_{p_4} is an $8''$. Then neither p_5 or even its parent (if it exists) can have another child without violating the order in which pixels are discovered when constructing the BFS tree. Hence, we have a 9-polyiamond that can be covered with two point guards and T_{p_5} is pixel-good.

(5c) T_{p_4} is an $8'$ and $|T_{p_5}| = 9$, 10, 11, or 12. Place two point guards on p_5: one at the vertex shared by p_5 and p_3 and another at the vertex shared by p_5 and p_3's sibling, p'_3. Then T_{p_4} and p_5 are covered. If p_5 has two children, then we consider the subtree of this other child, p'_4. Clearly, any path of length 3 or less from p_5 is necessarily covered since one of the two point guards is a vertex of this convex 3-polyiamond. This implies that $T_{p'_4}$ is covered by two point guards. At worst we have a 2:9 ratio and so T_{p_5} is pixel-good.

(5d) T_{p_4} is an $8'$, $T_{p'_4}$ has height 2 and $|T_{p_5}| = 13$. Then the guard placement in case (5b) will cover T_{p_5} since one of the guards is on the convex subpolyiamond(s) formed by p'_4 and its descendants. Therefore, T_{p_5} is pixel-good.

(5e) T_{p_4} is an $8'$, $T_{p'_4}$ has height 3 and $|T_{p_5}| = 13$. If p_5 has no parent, then we add a third point guard to the third vertex of p_5 so that all of T_{p_5} is covered. This gives us a 3:13 ratio on our final iteration. If p_5 has a parent and p_5 has no siblings, then we place a third guard as before and we obtain a 3:14 ratio so T_{p_6} is pixel-good. If p_5 has a sibling, p'_5 and $T_{p'_5}$ has height ≤ 2, then we may place a third guard as before and $T_{p'_5}$ will also be covered (see the left illustration in Figure 4.13). At worst, T_{p_6} has a ratio of 3:15 and is thus pixel-good. Else, we consider $T_{p'_5}$. The ways that $T_{p'_5}$ might not have a pixel-good subtree of $T_{p'_5}$ is if a.) $T_{p'_5}$ were a path of length 4 with p'_5 its head, b.) $T_{p'_5}$ were an $8'$, $8''$, or $8'''$ with p'_5 a center pixel, or $T_{p'_5}$ were equivalent in structure to T_{p_5}. In a.), if $T_{p'_5}$ is covered by the guards at the vertices of p_5, then we are done. Else we have a situation such as the one depicted on the right side of Figure 4.13. Here, a fourth guard is required, which we place at the third vertex of p_6. This results in a 4:18 = 2:9 ratio and so T_{p_6} is pixel-good. Both b.) and c.) are impossible to realize since p'_5 can have at most one child.

(5f) T_{p_4} is an $8'$ and $T_{p'_4}$ is a path of length 5. Then $T_{p'_4}$ can be covered with a single point guard and we place the other two guards as in case (5b), covering the rest of T_{p_5}. This gives us a 3:14 ratio and so T_{p_5} is pixel-good.

(5g) T_{p_4} is $8'$ and $T_{p'_4}$ is $8'$ or $8''$. The polyiamond can not be realized if $T_{p'_4}$ is $8''$. If $T_{p'_4}$ is $8'$, then the polyiamond can only be realized in one way and if we place 3 guards at the three vertices of p_5, T_{p_5} is covered. We then have a 3:17 ratio and so T_{p_5} is pixel-good.

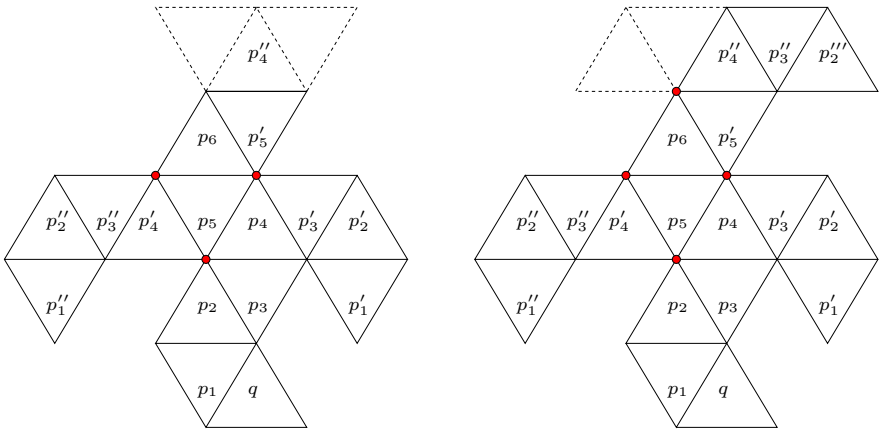


Figure 4.13: Left: The three point guards see all of T_{p_6} when T_{p_5} has height ≤ 2. Right: The four point guards see all of T_{p_6}. The dotted pixels on the right form the alternate path from p_5' requiring 4 guards.

Hence, we can inductively remove a subpolyiamond that is pixel-good at each iteration. For the base case, we may need an additional guard if one of the terminating situations mentioned in the above analysis arises (cases (5a) or (5e)) or, if we have between 1 and 4 pixels left over. This yields the desired bound of $\lfloor \frac{2m}{9} \rfloor + 1$ for all $m \geq 1$. Since the guards never needed to 'team-up' to cover a pixel, our bound holds in the all-or-nothing model as well. □

Corollary 4.3.11. *For a polyiamond P with $m \geq 6$ pixels and h holes, $\approx \frac{2m}{11}$ point guards are sometimes necessary while $\lfloor \frac{2m}{9} \rfloor + 1$ point guards are always sufficient to cover P, even in the all-or-nothing model.*

Pixel Guards

In the pixel guard case, we have analogous lemmas to the point guard version:

Lemma 4.3.12. *Let T be a BFS tree of the dual of a polyiamond P having a root node with degree ≤ 2 and height ≤ 4. Then T can be covered with a single pixel guard at the vertex corresponding to r.*

Proof. Consider any leaf node q in T. The unique subpolyiamond S formed by the path in G from q up to but not including r has between 0 and 4 pixels. Therefore, S is either empty, a single equilateral triangle, a rhombus,

a trapezoid, or a path of length 4. The first three possibilities are convex and since an edge of r is incident to S, they are covered. Otherwise S is a path of length 4. Consider the grandchild g of r in S. g and r share a vertex so the convex region formed by the pixels g to q, forming a trapezoid, are covered and so S is covered. □

Lemma 4.3.13. *Let P be an 10-polyiamond such that the dual of P is a path. Then there exists a subset D of P that require 2 guards (see Figure 4.14)*

Proof. We can think of P as being comprised of a central 2-polyiamond C with two subpaths, L and R (standing for left and right, respectively), of length 4 attached to the 2-polyiamond, one on each pixel. We can force a pixel guard to be at the 'right' pixel of C if and only if R has one of the two configurations shown in Figure 4.14 (left side). The 6 possible configurations of L (up to symmetry) that can not be completely covered by the pixel guard placed in the 'right' pixel of C are shown in Figure 4.14 (right side). □

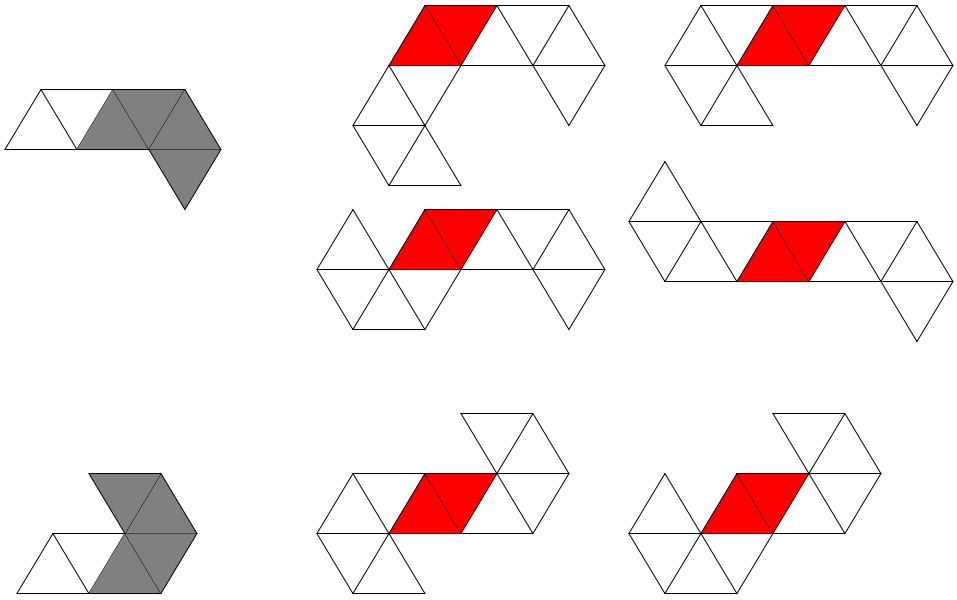


Figure 4.14: Left column: The two configurations of R (shaded) that require us to place a pixel guard at the right pixel of C; Middle and right columns: The 6 possible configurations of L, the left branches with four pixels, that force the 10-polyiamond to require 2 pixel guards (shaded).

Theorem 4.3.14. *For $m \geq 1$, $\lfloor \frac{3m}{16} \rfloor + 1$ pixel guards are always sufficient to cover a polyiamond P with m pixels and h holes, even in the all-or-nothing model.*

Proof. Compute a BFS tree T constructed from P's dual graph rooted at a vertex of degree ≤ 2. Let q be a lowest leaf of T, let p_1 be its parent, p_2 be its grandparent, p_3 be its great-grandparent, and generally p_i be its great-great-...-great-grandparent, where "great" is repeated $i-2$ times. Through case analysis, we will show that some rooted subtree of T_{p_6} is pixel-good. We begin with T_{p_4} and work our way up generations as needed.

T_{p_4} (and other subtrees of height ≤ 4): If $|T_{p_4}| \geq 6$, then since T_{p_4} can be covered with a single pixel guard placed at p_4 by Lemma 4.3.12, we have at worst a 1:6 ratio and so T_{p_4} is pixel-good. Else, T_{p_4} is a path of length 5 and we go up a generation.

T_{p_5} (and other subtrees of height ≤ 5): If any subtree of the children of T_{p_5} is pixel-good, we remove it and continue. Else, we consider several cases:

(5a) T_{p_5}'s dual is a path of length 6. Place a pixel guard at a central pixel. This guard covers T_{p_5} by Lemma 4.3.12 and so T_{p_5} is pixel-good.

(5b) $T_{p'_4}$ has height ≤ 2. Then the diameter of T_{p_5} is at most 9. By Lemma 4.3.12, a pixel guard placed at p_4 covers T_{p_5} with at worst a 1:9 ratio and so T_{p_5} is pixel-good.

(5c) T_{p_5} is a 10-polyiamond. If it is possible to cover T_{p_5} with 1 pixel guard we do so and T_{p_5} is pixel-good. Otherwise, T_{p_5} is one of the free polyiamonds depicted in Figure 4.14. If T_{p_5} has no parent, we place two pixel guards giving us a 1:5 ratio on the final iteration. Else, we go up a generation.

(5d) $|T_{p_4}| = 5$. Then T_{p_5} is an 11-polyiamond. If it is possible to cover T_{p_5} with 1 pixel guard we do so and T_{p_5} is pixel-good. Otherwise, we go up a generation.

T_{p_6} (and other subtrees of height ≤ 6): If any subtree of the children of T_{p_6} is pixel-good, we remove it and continue. Else, we consider several cases:

(6a) T_{p_5} is a (10) requiring 2 pixel guards and p_5 has no sibling. Then we have an (11) requiring two pixel guards and we go up a generation.

(6b) T_{p_5} is a (10) requiring 2 pixel guards and $T_{p_5'}$ has height ≤ 4. In this case, we place one pixel guard in p_5' and the other pixel guard in p_4. As we observe in Figure 4.15, these two guards still cover T_{p_5} (only one of the 4 cases is shown as they are all locally identical). The pixel guard at p_5' covers $T_{p_5'}$ and we have at worst at worst a 1:6 ratio so T_{p_6} is pixel-good.

(6c) T_{p_5} is an (11) requiring 2 pixel guards and p_5 has no sibling. Then we have a (12) requiring two pixel guards so T_{p_6} is pixel-good.

(6d) T_{p_5} is an (11) requiring 2 pixel guards and $T_{p_5'}$ has height ≤ 2. Then place pixel guards at p_4 and p_4' so that T_{p_5} and p_6 are covered. One of the pixel guards shares a vertex with p_5' so $T_{p_5'}$ is covered. We have at worst a 2:13 ratio so T_{p_6} is pixel-good.

(6e) T_{p_5} is an (11) requiring 2 pixel guards and $T_{p_5'}$ has height 3 or 4. Place pixel guards at p_4 and p_4' so that T_{p_5} is covered. Also, place a pixel guard at p_5' so that all of $T_{p_5'}$ is covered as well by Lemma 4.3.12. This gives us at worst a 3:16 ratio and so T_{p_6} is pixel-good.

(6f) T_{p_5} is an (11) requiring 2 pixel guards and $T_{p_5'}$ is a (10) or (11) requiring 2 pixel guards. Then we have a 2:11 or 4:23 ratio, either of which make T_{p_6} pixel-good.

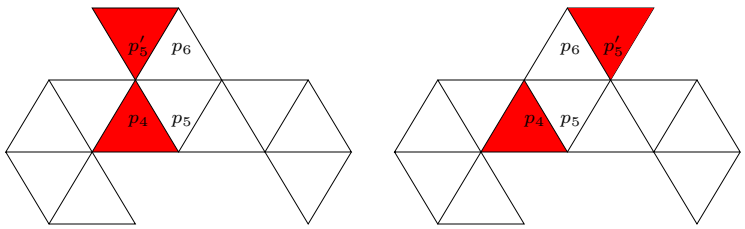


Figure 4.15: Illustrations of pixel guard placement for case (6b). Regardless of p_6's two possible configuration (left and right) the polyiamond defined by T_{p_5} is covered by the two pixel guards placed at p_4 and p_5'.

T_{p_7} **(and other subtrees of height ≤ 7):** If any subtree of the children of T_{p_7} is pixel-good, we remove it and continue. Else, we consider the following case: T_{p_6} is an (11) requiring 2 pixel guards and p_5 has no sibling. At worst

we require a 1:5 ratio to cover $T_{p_5'}$ so guarding T_{p_6} requires at worst a 3:16 and so T_{p_6} is pixel-good.

Hence, we can inductively remove a subpolyiamond that is pixel-good at each iteration. For the base case, we may need an additional guard if a terminating situation mentioned in the above analysis arises (case (5c)) or, if we have between 1 and 5 pixels left over. This yields the desired bound of $\lfloor \frac{3m}{16} \rfloor + 1$ for all $m \geq 1$. Since the guards never needed to 'team-up' to cover a pixel, our bound holds in the all-or-nothing model as well. □

Corollary 4.3.15. *For a polyiamond P with $m \geq 7$ pixels and h holes, $\approx \frac{m}{7}$ pixel guards are sometimes necessary while $\lfloor \frac{3m}{16} \rfloor + 1$ pixel guards are always sufficient to cover P.*

4.3.3 Polyhexes

Point Guards

Theorem 4.3.16. *$\lfloor \frac{m}{2} \rfloor$ point guards are sometimes necessary and always sufficient to cover an m-polyhex P, even in the all-or-nothing model.*

Proof. The lower bound is established through Figure 4.3. For the upper bound, compute a BFS tree rooted at any node in the dual graph of P. 2-color the resulting tree. Since no more than $\lfloor \frac{m}{2} \rfloor$ hexagons can be in the lesser color class, we cover the polyhex with at most $\lfloor \frac{m}{2} \rfloor$ point guards: one placed at the center of each hexagon in the lesser color class. □

Pixel Guards

We now investigate the pixel guard case for polyhexes. Before considering the unrestricted model, we have the following result:

Theorem 4.3.17. *For $m \geq 2$, $\lfloor \frac{m}{2} \rfloor$ pixel guards are sometimes necessary and always sufficient to cover an m-polyhex with h holes in the all-or-nothing model.*

Proof. The lower bound is established from replacing the point guards in Figure 4.2 (top) with pixel guards. The upper bound is established in the same way as the proof of Theorem 4.3.16. □

Construct a BFS tree rooted at a some vertex of degree ≤ 3. No special preference to the order in which children of a pixel are explored is required. Note that in the BFS tree, no vertex v can have more than 3 children since a 4th or 5th potentially adjacent vertex would also be adjacent to v's parent and hence already have been explored. As before, our approach will be to iteratively remove subtrees that are 'pixel-good' (have at worst a 3:7 ratio between pixel guards used and pixels covered). First, we introduce the following lemma:

Lemma 4.3.18. *There are 3 free 3-polyhexes and each requires one pixel guard. All 4-polyhexes require one pixel guard except $4'$ and $4''$, which each require 2 pixel guards.*

Proof. For the 3-polyhex case, pick a vertex of degree 2 in the dual and place a pixel guard there. Clearly this pixel and its two neighboring pixels are covered. For the 4-polyhex case, we consider all 7 free 4-polyhexes in Figure 4.16 and by inspection observe that each requires one guard, except $4'$ and $4''$, which each require 2 pixel guards. □

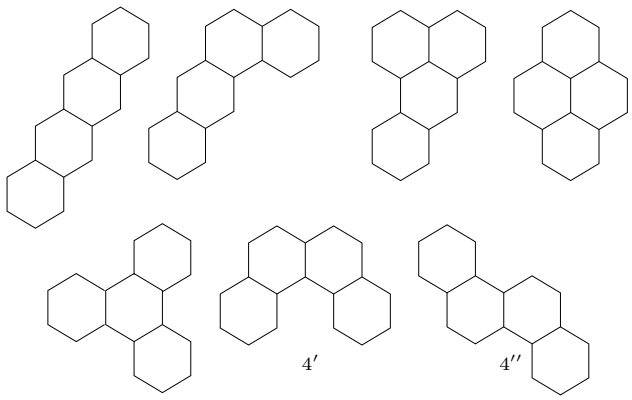


Figure 4.16: The free polyhexes with 4 pixels. All require one pixel guard except $4'$ and $4''$, which each require 2 pixel guards.

Theorem 4.3.19. *For $m \geq 1$, $\lfloor \frac{3m}{7} \rfloor + 1$ pixel guards are always sufficient to cover a polyhex P with m pixels and h holes.*

Proof. Again, we compute a BFS tree T constructed from P's dual graph rooted at a vertex of degree ≤ 3. Let q be a lowest leaf of T, let p_1 be its

parent, p_2 be its grandparent, p_3 be its great-grandparent, and generally p_i be its great-great-...-great-grandparent, where "great" is repeated $i-2$ times. Through case analysis, we will show that some rooted subtree of T_{p_3} is pixel-good. We begin with T_{p_1} and work our way up generations as needed.

T_{p_1} (and other subtrees of height ≤ 1): If $|T_{p_1}| = 3$ or 4, then a pixel guard placed at p_1 will cover p_1 and all its children. Hence, T_{p_1} is pixel-good. Else, p_1 has one child and we go up a generation.

T_{p_2} (and other subtrees of height ≤ 2): If any subtree of the children of T_{p_2} is pixel-good, we remove it and continue. Else, we consider several cases:

(2a) $|T_{p_2}| = 3$ or $|T_{p_2}| = 4$ and T_{p_2} is not $4'$ or $4''$. Then 1 pixel guard suffices by Lemma 4.3.18 and T_{p_2} is pixel-good.

(2b) $|T_{p_2}| = 5$ and p_2 has two children. Then T_{p_2} is a path of length 5 and can be covered with 2 pixel guards, one placed at each child of p_2 so that T_{p_2} is pixel-good.

(2c) $|T_{p_2}| = 5$ and p_2 has three children. Then placing a pixel guard at p_1 covers everything (p_2 and its three children form a star-shaped 4-polyhex and q is adjacent to p_1).

(2d) $|T_{p_2}| = 6$. We can imagine removing the child of p_2 that has no child, leaving us with a path of pixels that can be covered with 2 pixel guards placed at the other two children of p_2. The leaf we 'removed' is necessarily neighboring a pixel containing a pixel guard (assuming p_2 has a parent) so T_{p_2} is pixel-good. Even if p_2 has no parent, it turns out that the two pixel guards can 'team up' (in the assumed unrestricted visibility model) to cover the leaf child if it is not adjacent to either pixel containing the pixel guards.

(2e) $|T_{p_2}| = 7$. In this case a pixel guard placed at each child of p_2 suffices and so T_{p_2} is pixel-good. In fact, it can be shown that only 2 pixel guards are ever required if p_2 has a parent.

In the event that T_{p_2} is $4'$ or $4''$, we go up a generation and consider T_{p_3}.

T_{p_3} (and other subtrees of height ≤ 3): If any subtree of the children of T_{p_3} is pixel-good, we remove it and continue. Else, we consider several cases:

(3a) p_3 has only one child. Then we can simply place pixel guards at p_1 and p_2 so that T_{p_3} is pixel-good.

(3b) p_3 has two children and one child is a leaf. Then we can cover T_{p_3} by placing pixel guards at p_1 and p_3 since the subpolyhex formed by p_1, p_2, p_3 and p_1', p_1's sibling, is covered by these two pixel guards. In particular, although neither of the pixel guards at p_1 or p_3 may be able to cover p_1' by itself, they can 'team up' (in the assumed unrestricted visibility model) to cover p_1' (see Figure 4.17).

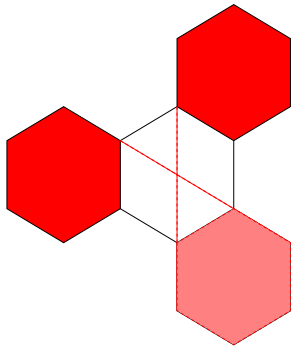


Figure 4.17: Although neither of the dark shaded pixel guards (representing pixel guards at p_1 and p_3 in case (3b)) can cover the light shaded pixel by itself, together they cover it.

(3c) p_3 has two children and one of them has only a single leaf child itself. Here, $|T_{p_3}| = 7$ and T_{p_3} can be covered by placing 3 pixel guards at p_1, p_2 and the sibling of p_2. T_{p_3} is then pixel-good.

(3d) p_3 has two children and both of them are roots of a $4'$ or $4''$. In this case, we place pixel guards at p_3, p_1 and p_1'', the cousin of p_1 that is not a leaf. As in case (3b), we can be assured that all pixels including the siblings of p_1 and p_1'' are covered.

(3e) p_3 has 3 children and two of them are leaves. A pixel guard placed at p_3 covers itself and its two leaf children. Another pixel guard placed at p_1 covers the rest as in case (3b) so T_{p_3} is pixel-good.

(3f) p_3 has 3 children, one of which is a leaf child and another that has a single leaf child itself. Place pixel guards at p_1, p_3 and p_2'', the sibling of p_2 that has a child. Then we have a 3:8 ratio and T_{p_3} is pixel-good.

(3f) p_3 has 3 children and two of them have a single leaf child so that $|T_{p_3}| = 9$. Here, we place 3 pixel guards at p_1 and the two siblings of p_2: p_2' and p_2''. Clearly, these three pixels are covered along with q, the children of both p_2' and p_2'', and p_3. Also, assuming that p_3 has a parent, one of p_2' and p_2'' must be adjacent to p_2 so that p_2 is covered. We can then use an argument identical to case (3b) to show that p_1's sibling is covered as well. Hence, T_{p_3} is pixel-good. If p_3 doesn't have a parent then we may require 4 guards, giving us a 4:9 ratio.

(3g) p_3 has 3 children and each is a root of a $4'$ or $4''$. This case isn't realizable if p_3 has a parent. If p_3 does not have a parent, then only 4 pixel guards are necessary if we place them at p_3, p_1 and the cousins of p_1 that have a child. Same analysis from case (3b) applies and T_{p_3} is pixel-good.

Hence, we can always remove a subpolyhex that is pixel-good at each iteration except perhaps the final iteration where we may require an additional pixel guard. This yields the desired bound of $\lfloor \frac{3m}{7} \rfloor + 1$ for all $m \geq 1$. □

Corollary 4.3.20. *For a polyhex P with $m \geq 3$ pixels and h holes, $\approx \frac{5m}{14}$ pixel guards are sometimes necessary while $\lfloor \frac{3m}{7} \rfloor + 1$ pixel guards are always sufficient to cover P.*

4.4 Conclusion

A collection of combinatorial results are given for guarding polycubes, polyiamonds and polyhexes. The bounds apply to both simple and non-simple polyforms. It would be interesting to see if any of the loose bounds could be tightened. Also, since polyominoes and polycubes have the same combinatorial bounds using point guards, it is natural to ask if this result extends to *polyhypercubes* in $d \geq 4$ dimensions.

Chapter 5

Spiral Serpentine Polygonization of a Planar Point Set[1]

5.1 Introduction

A polygonization of a planar point set S is a simple polygon having S as the set of its vertices. Different types of polygonizations have been investigated in settings where objects are being constructed from limited data, such as pattern recognition and image reconstruction [1], [18], [21]. The number of polygonizations for a given point set can be exponential in n, even when restricted to monotone or star-shaped polygonizations [38].

The existence of a polygonization for any point set S in general position was established by Steinhaus [48]. The Euclidean-TSP on S was shown to be simple by Quintas and Supnick, yielding the existence of a polygonization for any planar point set S [44]. Graham demonstrated that a *star-shaped* polygonization could be constructed explicitly in $O(n \log n)$ time [25]. Later, Grünbaum showed implicitly that every point set S, where not all points are collinear, has a *monotone* polygonization and that it can be computed in $O(n \log n)$ time [26]. Agarwal et al. have discussed the attractiveness of the subset of polygonizations that admit *thin* triangulations, which minimize the number of

[1]This chapter is based on work joint with Joseph S. B. Mitchell. A preliminary version appeared in *Proceedings of the XIV Spanish Workshop on Computational Geometry (2011)* [31] and a full version was invited to appear in *Lecture Notes in Computer Science: The Ferran Hurtado 60^{th} Birthday Festschrift* [33].

nodes of degree three in the dual, and in particular, *serpentine* triangulations, whose dual graph is a path. In [3], the authors gave an $O(n \log n)$ algorithm for computing a monotone serpentine polygonization of a point set S of n points, not all of them collinear.

The *reflexivity* of a point set S was introduced and studied by Arkin et al. [5]; the reflexivity of S is the minimum number of reflex vertices possible among all polygonizations of S. This paper motivated us to consider the problem of determining the *inflection number* of a point set S, which we define to be the minimum number of transitions between reflex and convex vertices possible among all polygonizations of S. We show that one can always polygonize S with zero or two transitions; the inflection number is zero if and only if S is in convex position.

In particular, we demonstrate that any point set S has a *spiral* serpentine polygonization. A spiral serpentine polygon is a simple polygon possessing at most one chain of reflex vertices, exactly one chain of convex vertices (see Figure 5.1) and admitting a serpentine triangulation. We note that it is trivial to find a spiraling polygonal simple *path* through a set of n points in the plane as in Figure 5.2. The task here is to construct a simple *cycle* through the points that possesses the spiral and serpentine properties.

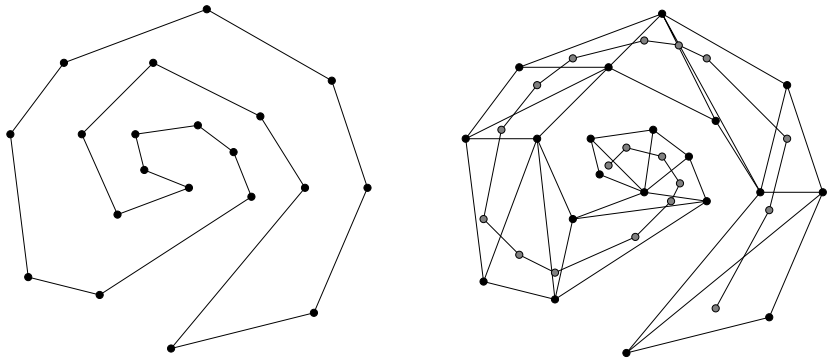


Figure 5.1: A spiral polygon is shown in (a.) and a serpentine triangulation of this polygon is shown in (b.), where the dual of the triangulation is the path depicted.

We present a simple algorithm in Section 5.2 for constructing such a polygonization in $O(n \log n)$ time, requiring $O(n)$ space and explicitly giving a serpentine triangulation at no extra cost. Both the run-time and space complexities are optimal. To establish that the runtime is $\Omega(n \log n)$, we can use

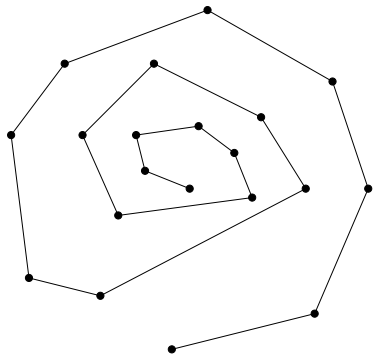


Figure 5.2: A spiraling polygonal simple path is easy to construct.

the same approach as in [46] where the input S contains $n-1$ collinear points with one additional point p not collinear with the rest. If we rotate S so that p has the lowest y-coordinate among all the points, then our algorithm will give a unique spiral serpentine polygonization of the point set that can be used to sort the $n-1$ collinear points.

In Section 5.3, we establish the correctness of the algorithm. Section 5.4 gives examples computed from an implementation of the algorithm. Section 5.5 provides a closing discussion and some related open questions.

5.2 The Algorithm

Here we introduce the algorithm `SpiralSerpentinePolygonize`, which produces a spiral serpentine polygonization of a planar set S of $n \geq 3$ points. Refer to the pseudocode below.

In the first step of the algorithm, we compute the convex hull, H, of S and initialize a semi-dynamic convex hull data structure that allows only deletion operations[2] [28]. Three arrays are also initialized: C, storing the convex chain, R, storing the (possibly empty) reflex chain, and D, storing the set of diagonals for the triangulation.

After establishing the first two vertices, u and v, of the convex chain C in step 2.) and determining the next point w of S to consider in step 3.),

[2]In a previous extended abstract of this paper [31], we give an alternative to using the dynamic convex hull data structure, based on computing the *convex layers* [15] of all points in S and exploiting special structure in the layers when our algorithm requires convex hull queries.

the algorithm enters a **while** loop in step 4.) that processes one remaining point per iteration. In particular, at the start of each iteration of the loop we check whether or not the (possibly unbounded) wedge Q formed by rays emanating from w along the directions $\overrightarrow{uw}$ and $\overrightarrow{vw}$ contains any points of H (see Figure 5.3). This can be determined in $O(\log n)$ time, using binary search in the data structure storing the points H.

If Q has a point of H, the first point q encountered by rotating counterclockwise from the ray centered at w in the direction $\overrightarrow{uw}$ is located (in time $O(\log n)$). (In determining q, ties are again broken by picking the point closest to w.) We then update u to be w and w to be q, append u to the reflex chain R, and append (u, v) to the set of diagonals D.

Otherwise, the first point q encountered by rotating counterclockwise the ray centered at u in the direction $\overrightarrow{uw}$ is located. We then update v to be w and w to be q, append v to the convex chain C, and append (u, v) to the set of diagonals D.

Once H is empty, we append w to C and if $n > 3$, (u, v) to D. We let the polygonization P be stored as an array containing the concatenation of R in reverse order to the end of C. The polygonization can be constructed by outputting the edges $(P[i], P[i+1])$, $(0 \leq i \leq n-2)$, along with $(P[n-1], P[0])$. The triangulation T is constructed via the polygonization P and set of diagonals D.

Algorithm: `SpiralSerpentinePolygonize`(S)

1.) Initialize H to be $\mathcal{CH}(S)$, stored in a dynamic (deletion only) convex hull data structure. Initialize the arrays C and R for the convex and reflex chains, respectively, and the array of vertex pairs D for the set of diagonals, to be empty: $C = R = D = \emptyset$.
2.) Determine u, the vertex of H with minimum y-coordinate (breaking ties by maximizing x-coordinate) and append it to C. Let v to be the next point on H in the counterclockwise direction and append it to C. Delete u and v from H.
3.) Let w be the first point encountered by rotating counterclockwise the ray emanating from u and passing through v (breaking ties by picking the point closest to u). Delete w from H.
4.) **while** H is not empty
    Let Q be the wedge formed by rays emanating from w along the directions $\overrightarrow{uw}$ and $\overrightarrow{vw}$.
    if $Q \cap H \neq \emptyset$
        Find the next $q \in H$ encountered by rotating ccw the ray centered at w in the direction $\overrightarrow{uw}$.
        Set $u = w$
        Set $w = q$
        Append u to R
        Append (u, v) to D
        Delete q from H
    else
        Find the next $q \in H$ encountered by rotating ccw the ray centered at u in the direction $\overrightarrow{uw}$.
        Set $v = w$
        Set $w = q$
        Append v to C
        Append (u, v) to D
        Delete q from H
end while
5.) Append w to C and (u, v) to D. Let P be the array obtained by concatenating R in reverse order to the end of C. The polygonization can be constructed by outputting the edges $(P[i], P[i+1])$, $(0 \leq i \leq n-2)$, along with $(P[n-1], P[0])$. The triangulation T is constructed via the polygon P and diagonals D.

We now summarize the important invariants of the algorithm.

a.) Except for the final point appended in step 5.), each point w is either relabeled v and appended to the convex chain C or relabeled u and appended to the reflex chain R.

b.) In step 4.), w (once relabeled u or v) always belongs to one single triangle t of the current triangulation, T_k. Therefore, t is always a leaf node in the dual graph (a path) of T_k.

c.) H is always contained in the intersection of left half-planes of edges along the reflex chain in the counterclockwise direction and the directed line $\overline{uw}$. This region is the union of the light shaded region and Q, as depicted in Figure 5.3.

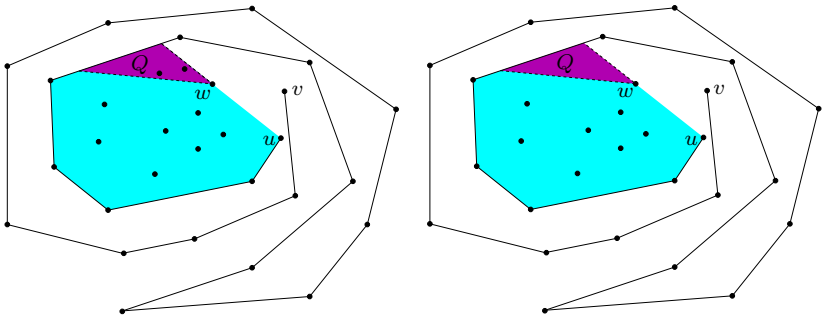


Figure 5.3: The two possible cases that arise during each iteration of the algorithm's **while** loop: When the dark shaded region Q contains at least one point of H (left) and when it contains no points of H (right).

5.3 Correctness

Lemma 5.3.1. `SpiralSerpentinePolygonize` *constructs a spiral polygon* P.

Proof. The proof is by induction on the iteration counter. After the first iteration of the **while** loop, we have just one triangle, which is trivially spiral. Assume the claim holds after $k < n$ iterations of the **while** loop and consider the state after the $(k+1)^{th}$ iteration. We remove the point $w = q_{k+1}$ that was

most recently discovered along with the two edges incident to q_k, the point discovered on the k^{th} iteration. Use labels u, w, v as assigned at the end of the k^{th} iteration. The resulting polygon is spiral by the induction hypothesis. We have two cases to consider: when q_{k+1} is in Q_{k+1} (the wedge Q during the $(k+1)^{th}$ iteration) and when it is not.

We first assume that q_{k+1} is in Q_{k+1}. In this case we wish to show that when q_{k+1} is processed, (a) v remains a convex vertex, and (b) w becomes a reflex vertex. Let v' be the convex vertex adjacent to v in the clockwise direction. We observe that uvq_{k+1} must form a left turn (otherwise, q_{k+1} is not in Q_{k+1}). Consider the wedge formed by rays centered at v along the directions $\overrightarrow{uv}$ and $\overrightarrow{v'v}$ (see Figure 5.4), which was previously examined after the vertex currently labeled v was inserted into C. The subsequent point discovered, w, was not in this wedge and so the wedge is empty. It follows then that q_{k+1} must be to the left of the directed line $v'v$ so that v remains a convex vertex. Since q_{k+1} is in Q_{k+1}, uwq_{k+1} must form a left turn, which implies that w becomes a reflex vertex.

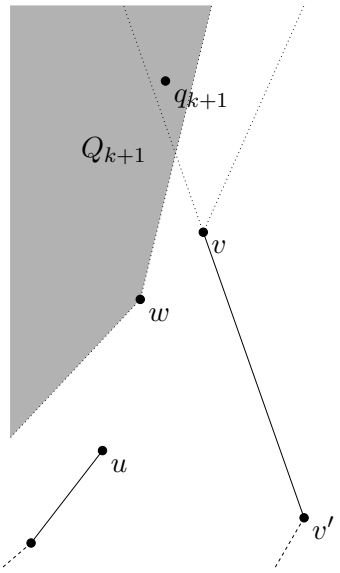


Figure 5.4: $v'vq_{k+1}$ cannot form a right turn.

Next we suppose that q_{k+1} is not in Q_{k+1}. Here, we show that when q_{k+1} is processed, (a) u retains its convexity (convex or reflex), and (b) w becomes convex. Consider u being reflex in the k^{th} iteration of the loop. Then on the $(k+1)^{th}$ iteration the ray $\overrightarrow{uw}$ is rotated counterclockwise until it encounters

q_{k+1} in H. Clearly, u's interior angle cannot decrease so it remains reflex. Next we consider u being convex in the k^{th} iteration. Here, u represents the point first added to C in step 2.) of the algorithm. Since $u \in \mathcal{CH}(S)$, u must remain convex after the $(k+1)^{th}$ iteration. Finally, since q_{k+1} is not in Q_{k+1}, vwq_{k+1} necessarily forms a left turn, indicating that w remains convex. □

Lemma 5.3.2. `SpiralSerpentinePolygonize` *constructs a triangulation of* P.

Proof. In each iteration of the **while** loop of the algorithm, a triangle is effectively appended to an edge of the polygonization constructed so far. Specifically, a point w is relabeled as u or v and segments (one boundary edge and one diagonal) are attached from this vertex to the most recently added vertices along the convex and reflex chains, C and R, respectively. (If R is empty, the second segment connects to the first point selected in S in step 2.)) By algorithm invariant c.), w is to the left of all previously constructed edges along the reflex chain in the counterclockwise direction and the directed line $\overline{uw}$. It follows that the new segments appended do not intersect any other boundary points of the polygonization constructed so far, yielding a valid updated triangulation. □

Lemma 5.3.3. *The triangulation constructed by* `SpiralSerpentinePolygonize` *is serpentine.*

Proof. After the first iteration we have a single triangle, which is trivially serpentine. During each subsequent iteration, the new triangle formed is adjacent to a leaf of the dual path of the current triangulation constructed so far. Hence, the updated triangulation remains serpentine. □

Lemmas 5.3.1, 5.3.2 and 5.3.3 yield the desired result, stated in the following theorem:

Theorem 5.3.4. `SpiralSerpentinePolygonize` *constructs a spiral serpentine polygon.*

Finally, we examine the algorithm's runtime and space usage:

Theorem 5.3.5. `SpiralSerpentinePolygonize` *runs in* $O(n \log n)$ *time and requires* $O(n)$ *space.*

Proof. The semi-dynamic convex hull data structure requires $O(n \log n)$ preprocessing time, $O(\log n)$ amortized deletion time and $O(n)$ space [28]. During each iteration of the **while** loop, a point of S is processed, yielding at most $n - 3$ iterations. For each iteration of the while loop, it takes $O(\log n)$ time to determine if Q contains a point of H, $O(\log n)$ time to find the next vertex q, and constant time to update the arrays storing one of the polygonal chains (C or R) and the diagonals (D). This gives us the desired $O(n \log n)$ run time, which is optimal. Since the semi-dynamic convex hull structure uses $O(n)$ space and the input points and the outputted polygon and set of diagonals (for the triangulation) can be stored in arrays of linear size, we require just $O(n)$ space. □

5.4 Implementation and Examples

We have implemented `SpiralSerpentinePolygonize` in Java[3]. In Figure 5 we display three different input point sets, along with corresponding spiral polygonizations and serpentine triangulations produced by our algorithm.

In the first row of Figure 5, our input set is a cloud of 25 random points. The second row depicts a "dumbbell shaped" point set, and the last row resembles the face of Mickey Mouse. In each row, we show the point set (left), the spiral polygonization given by the algorithm (middle), and the serpentine triangulation of the spiral polygonization (right). Note that the spiral polygonization is not a good curve reconstruction polygonization; the spiral polygonizations we compute are not indicative of the "shape" of the point sets.

5.5 Conclusion

We have shown that every planar point set S with $n \geq 3$ points admits a spiral serpentine polygonization, which can be computed with its accompanying triangulation in optimal $O(n \log n)$ time and $O(n)$ space.

This work suggests some open questions for further study. Deneen and Shute [18] have investigated the combinatorics of star-shaped polygonizations. An analogous question is how many spiral polygonizations exist among all

[3] An interactive Java applet can be accessed at `http://www.ams.stonybrook.edu/~jsbm/SpiralSerpentinePolygonize.html`

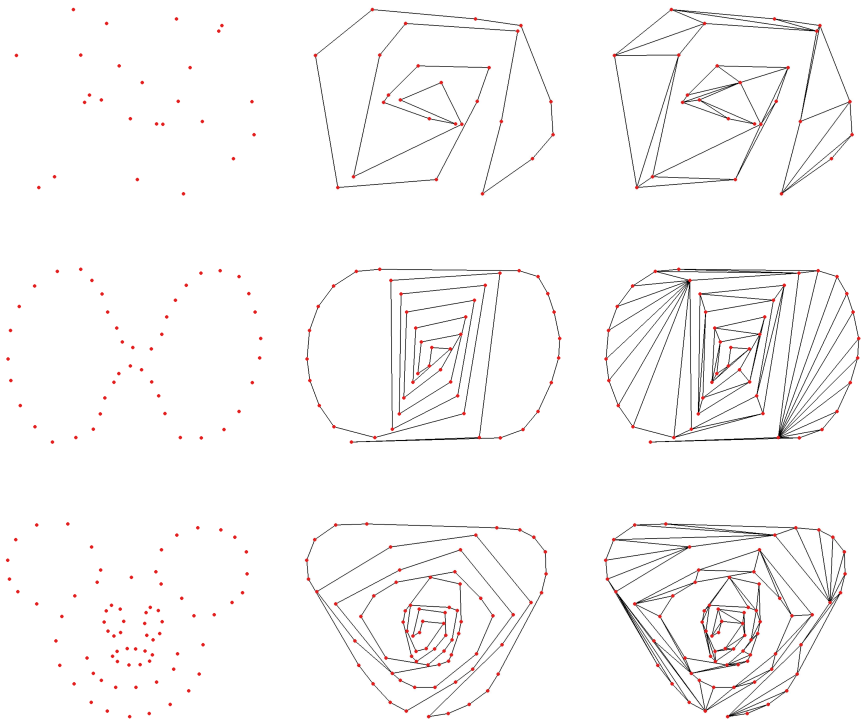


Figure 5.5: Left to right in each row: an input point set S, a spiral polygonization of S produced, and a serpentine triangulation of the spiral polygonization.

sets of n points in the plane? Secondly, what is the complexity of finding a minimum length spiral serpentine polygonization or a minimum weight serpentine triangulation? Finally, in 3D we pose a surface reconstruction problem: what is the minimum number of transitions between *convex patches* and *reflex patches* based on dihedral angles?

Chapter 6

Beacon-Based Routing and Coverage[1]

6.1 Introduction

The model of beacon-based routing in this paper is an analog of geographical greedy routing in sensor networks in the continuous setting. In geographical routing [12, 35], each node is given a Euclidean coordinate and a message is delivered to the neighbor whose Euclidean distance to the destination is the smallest. When sensor distribution is very dense (i.e., close to infinity), geographical routing will always follow the straight line towards the destination, or, when the message hits the network boundary, may follow a boundary edge to greedily minimize the distance to the destination. This is precisely the model of beacon-based routing in this paper, where the destination is a beacon.

Our model is also related to a family of routing schemes in sensor networks that use landmarks [20, 22, 39]. A subset of nodes, called landmarks, first flood the entire network such that each node records the distance to each landmark. For routing towards a destination, a function based on the distance vector to the landmarks is used to select the next hop. The one most similar to our model is adopted in [39]. In [39], the message is routed towards one landmark

[1]This chapter is based on work joint with Michael Biro, Jie Gao, Irina Kostitsyna and Joseph S. B. Mitchell. Preliminary versions appear in the *21st Fall Workshop on Computational Geometry 2011* [9] and *1st Computational Geometry: Young Researchers Forum 2012* [10].

until the current node is equal distance away from the landmark as the destination. At this point another landmark is selected. The paper shows that by carefully choosing the landmarks, the routing path is within a constant factor of the shortest path. In this paper we examine the combinatorial structures for landmark placement, to support this type of routing.

In our model, a beacon can occupy a point location on the interior or the boundary of P, ∂P. When a beacon is *activated*, all points $p \in P$ move along straight lines toward b until they either reach b or make contact with ∂P. If contact is made with ∂P, p will follow along ∂P as long as its straight line distance to b decreases monotonically. p may alternate between moving in a straight line path toward b on the interior of P and following along ∂P. If p is unable to move so that its distance to b decreases monotonically, we say p is 'stuck' and has reached a local minimum or *dead point* on ∂P (see Figure 6.1). If p reaches b we say that b *attracts* p. Two points are *routed* if there is a sequence of beacons that can be activated and then deactivated, one at a time in order, such that a point beginning at a source s would visit each beacon in the sequence after it is activated and terminate at a destination t, which we always assume to be a beacon itself.

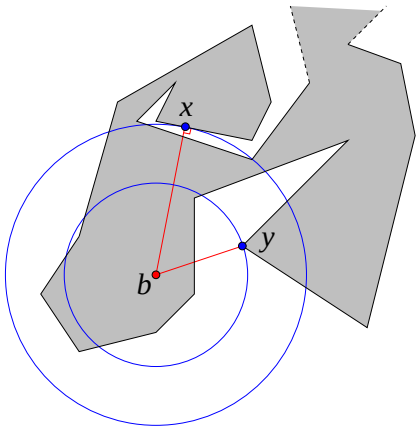


Figure 6.1: x and y are dead points with respect to the beacon, b.

6.1.1 Our Results

We first present results pertaining to beacon-based routing (Section 6.2). We show that $\lfloor \frac{n}{2} \rfloor - 1$ beacons are sometimes necessary and always sufficient to

route between any pair of vertices in a simple polygon P. We show that it is NP-hard to find a minimum cardinality set of point beacons to route from all points $s \in P$ to a given destination point t. Using an algorithm to compute the *attraction region* of a beacon b (the locus of points in P that can reach b when it is activated), we establish a polynomial time algorithm for routing from a point s to a point t using a discrete set of candidate beacons.

Next, we investigate the complexity of beacon coverage (Section 6.3). Finding a minimum sized guard set to cover a simple polygon is known to be NP-hard [37]. We show that finding a minimum cardinality set of beacons B to cover a simple polygon P is also NP-hard.

6.2 Beacon-Based Routing

We begin with a combinatorial result for beacon-based routing:

6.2.1 Combinatorics of All Pairs Routing in a Simple Polygon

Theorem 6.2.1. $\lfloor \frac{n}{2} \rfloor - 1$ *beacons are sometimes necessary and always sufficient to route between any pair of vertices in a simple polygon* P.

Proof. We can see from Figure 6.2 that $\lfloor \frac{n}{2} \rfloor - 1$ beacons are sometimes necessary.

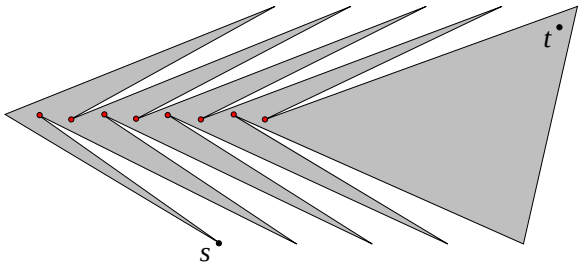


Figure 6.2: $\lfloor \frac{n}{2} \rfloor - 1$ beacons are sometimes necessary to route between all pairs of points in a simple polygon. Here, $n = 19$ and 8 beacons (light filled circles) are required to route from s to t.

To establish the upper bound, we first triangulate P and construct the dual graph G of the resulting triangulation. Beginning with a leaf node of G, we begin to peel off pairs of adjacent triangles. Let's call the two triangles σ_1 and σ_2, where σ_1 is the leaf triangle. We place a beacon at one vertex of the common edge of the two triangles and argue that one can navigate from any point p in either triangle to the beacon, or from the beacon to p, using *greedy routing*. We conduct case analysis on the number of triangles adjacent to σ_2 other than σ_1:

(i) σ_2 has only one additional adjacent triangle σ_3. Suppose $\sigma_1 = \triangle ABC$, $\sigma_2 = \triangle BCD$. σ_3 is then either $\triangle BDE$ or $\triangle CDE$. If $\sigma_3 = \triangle CDE$, then we place a beacon b at the vertex C and otherwise we place b at B. In either case, since b is contained in each of the three triangles, any point p in these three triangles can navigate to b and vice-versa (see Figure 6.3 (i)).

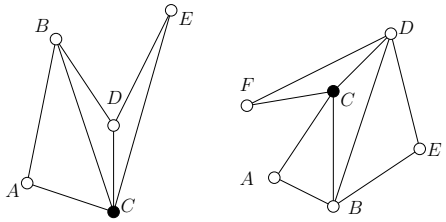


Figure 6.3: (i) The beacon b is placed at a vertex common to three triangles; (ii) The beacon b is placed at vertex C since $\angle FCB > 3\pi/2$. Any point in the four triangles can then navigate to b and vice-versa without any additional beacons needed.

(ii) σ_2 has two additional adjacent triangles σ_3, σ_4. Assume that $\sigma_1 = \triangle ABC$, $\sigma_2 = \triangle BCD$, $\sigma_3 = \triangle BDE$, $\sigma_4 = \triangle CDF$. We place a beacon b at C if $\angle FCB > 3\pi/2$, or at B if $\angle EBC > 3\pi/2$. We note that the two conditions cannot simultaneously be true. In particular, if $\angle FCB > 3\pi/2$, then $\angle BCD$ must be obtuse. If $\angle EBC > 3\pi/2$, then $\angle CBD$ must be obtuse. $\triangle BCD$ cannot have two obtuse angles. If neither of these conditions hold, then we place b arbitrarily at either B or C. Now, assume that we place b at vertex C. Then it must be the case that $\angle EBC \leq 3\pi/2$. Therefore, all points inside $\triangle BDE$ can reach b

and vice versa. Also as C is a vertex of σ_1, σ_2 and σ_4, all points inside theses three triangles can reach b and vice versa. Hence, the claim is true (see Figure 6.3 (ii)).

Given the basic steps as shown above, we will place beacons in a recursive manner: We take any leaf triangle σ_1 of the triangulation of P and place a beacon at one of the vertices of the shared edge of the pair σ_1 and its adjacent triangle, σ_2.

1. If P is a single triangle, we do nothing. If P has at most one more triangle besides σ_1 and σ_2 or P has only two more triangles but both are adjacent to σ_2, then we are done (see Figure 6.4 (i)). By the above arguments we can navigate from any source point to any destination point by using the single beacon: First route from the source to the beacon and then route from the beacon to the destination (which is always a beacon).

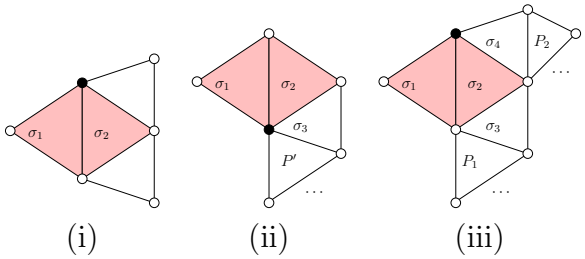


Figure 6.4: Inductive placement of beacons. (i): Base case; (ii): Peeling off σ_1 and σ_2 leaves a simple polygon; (iii): Peeling off σ_2 partitions P into two pieces, P_1 and P_2

2. Otherwise, we peel off σ_1 and σ_2. There are two subcases to consider:

   (a) σ_2 is only adjacent to one more triangle σ_3 (i.e., σ_2 has degree 2 in the dual graph; see Figure 6.4 (ii)). In this case peeling off σ_1 and σ_2 will still leave a simple polygon P'. We can recursively 'beaconize' P'. Now we argue that one can navigate with the union of these beacons. In particular, if the source and destination pair are both in $\sigma_1 \cup \sigma_2$ or both in P', then we can navigate by induction hypothesis. If the source and destination pair are separated in $\sigma_1 \cup \sigma_2$ and P', we can use the beacon x of σ_2 and the beacon y of σ_3 (if it is a

different beacon) to help guide the message across the two pieces. By the analysis of the basic case, any point inside σ_3, in particular, the beacon y, is reachable to and from the beacon x in σ_2. Thus navigation works in this case.

(b) σ_2 is adjacent to two other triangles σ_3 and σ_4. Thus peeling off σ_1 and σ_2 will partition the triangulation to two pieces, say P_1 and P_2. Suppose that P_1 contains σ_3 and P_2 contains σ_4 (see Figure 6.4 (iii)). By the same argument as above, we can use beacon x in σ_2 to navigate between the three pieces $\sigma_1 \cup \sigma_2$, P_1, and P_2.

With the algorithm, we can see that each time we peel off two triangles at a time and place one beacon. Thus the total number of beacons we place would be at most $\lfloor \frac{n-2}{2} \rfloor = \lfloor \frac{n}{2} \rfloor - 1$. □

In the subsequent theorem, we establish the hardness of all source routing in a simple polygon.

Theorem 6.2.2. *It is NP-hard to find a minimum cardinality set of beacons to route from all source points s to a given destination point t in a simple polygon.*

Proof. We prove hardness by reducing from the LINE HITTING problem: given an arrangement of n lines in the plane, place a minimum cardinality set of points S so that each line intersects ('hits') at least one point of S. Given an instance of the LINE HITTING problem, we construct a 'spike box' large enough for its rectangular body to contain a positive length segment of each line and all intersection points formed by the arrangement. The spike box contains a protruding zigzag spike gadget for each line in the arrangement. We let the destination point t be any point in the body of the spike box that is not on any line and let there be a source point s at the end of each spike. A zigzag spike gadget (see Figure 6.5) is constructed at either of the two locations a line exits the body of the spike box so that activating a beacon anywhere in the dark grey region $G(s)$ attracts s at the top of the zigzag spike to the beacon. Note that the light grey regions in the spike are regions in which an activated beacon could attract a point down from the above left and right hand horizontal edges, but not down the slanted edges. Since each beacon must be activated once in a sequence, the most efficient way to route from s in a zigzag spike to the body of the spike box is clearly by placing one beacon in $G(s)$ (else we would need to use multiple beacons to accomplish this task).

We can make each region $G(s)$ as narrow as we please so we will treat these regions as line segments.

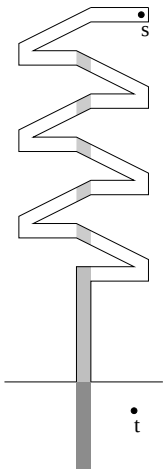


Figure 6.5: A zigzag spike gadget

Since t is in the body of the spike box, t can attract any other point in the body, with no other beacons required. Since t is not on any line of the arrangement, t is not in $G(s)$. Therefore, we need a beacon in each $G(s)$ to route from s to the body of the spike box.

Given an arrangement of lines that can be covered with k points, we can route to t with k beacons by placing the beacons on their corresponding lines. Since each line is covered by a point, there is a point in each $G(s)$ and so every point can be routed to t. If we can route to t using $k < n$ beacons, then we can cover the lines with k points. Since k is small, there must be a beacon in each $G(s)$. Place the points on the lines corresponding to the beacons, and there will be a point hitting every line. □

We now show the hardness of routing from a particular source point s to any destination point t in a simple polygon.

Theorem 6.2.3. *It is NP-hard to find a minimum cardinality set of beacons to route from a particular source point s to any destination point t in a simple polygon.*

Proof. We again reduce from the Line Hitting problem. The reasoning

is nearly identical to the proof of Theorem 6.2.2 using the gadget found in Figure 6.6 so we omit the details. □

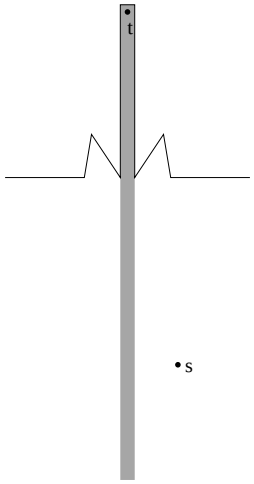


Figure 6.6: A spike gadget used for showing that all destination routing from a given source point is NP-hard

6.2.2 Computing the Attraction Region of a Beacon

In this section we devise an algorithm for computing the attraction region of a beacon that runs in $O(n \log n)$ time and uses $O(n)$ space.[2] We will consider the case in which P is simple first and then generalize to the case where P has polygonal holes. We begin with a survey of relevant properties of dead points.

Dead point properties

Definition 6.2.4. *For each dead point $d \in D(b)$, define the* dead region *of d, $R(d)$, to be the locus of points that reach d if the beacon at b is activated.*

Lemma 6.2.5. *All dead points lie on the boundary of P, ∂P.*

[2]Biro et al. have recently improved the running time for computing the attraction region of a beacon in a simple polygon to $O(n)$ using a triangulation of the polygon [10].

Proof. Assume that p is a point not in ∂P. Then, there exists a ball centered at p contained entirely in P. When b is turned on, p is unconstrained and will move a finite distance towards b. Therefore, p is not a dead point. □

Since ∂P is a sequence of line segments, we define a *vertex dead point* as a dead point lying on a vertex of the polygon and we define an *edge dead point* as a dead point that lies on the interior of an edge of the polygon.

Lemma 6.2.6. *If d is a dead point then the line $\overline{bd}$ crosses ∂P an even number of times.*

Proof. Take a point p such that $\overline{bp}$ crosses ∂P an odd number of times. The boundary of P divides the line $\overline{bp}$ into segments. The segments alternate from being inside the polygon to being outside the polygon. Since $b \in P$, the first segment is inside the polygon, and therefore, all odd segments lie inside the polygon. Since $\overline{bp}$ crosses ∂P an odd number of times, the final segment is odd and lies inside the polygon. When b is activated, p is free to move along that final segment of line $\overline{bp}$ and so p is not a dead point. □

Lemma 6.2.7. *A point d on the interior of an edge e of P is an edge dead point if and only if $\overrightarrow{bd} \perp e$ and $\overline{bd}$ crosses ∂P an even number of times.*

Proof. Take a point q on an edge $e = (p_i, p_{i+1})$ such that $\overrightarrow{bp} \not\perp \overrightarrow{p_i p_{i+1}}$. Then the angle between the two vectors is either greater than, or less than, 90°. Suppose without loss of generality that it is more. Then, when b is activated, q will be able to move towards b, either along $\overrightarrow{bq}$ or along $\overrightarrow{p_i p_{i+1}}$, depending on whether e is a near or far edge. Therefore q is not a dead point. This and Lemma 6.2.6 imply that if d is an edge dead point then $\overrightarrow{bd} \perp e$ and $\overline{bd}$ crosses ∂P an even number of times.

If $\overrightarrow{bd} \perp e$ and $\overline{bd}$ crosses ∂P an even number of times, then d is constrained to move along e, and any movement along e moves d farther from b. Therefore d is an edge dead point. □

Lemma 6.2.8. *If p_{i-1}, p, p_{i+1} is a sequence of vertices of the polygon, p is a vertex dead point if and only if $\overrightarrow{bp} \cdot \overrightarrow{pp_{i-1}} > 0$ and $\overrightarrow{bp} \cdot \overrightarrow{pp_{i+1}} > 0$ and $\overline{bp}$ crosses ∂P an even number of times.*

Proof. Suppose not, say $\overrightarrow{bp} \cdot \overrightarrow{pp_{i-1}} < 0$. Then, the angle $bpp_{i-1} < 90°$ and so p may move toward b along $\overrightarrow{pp_{i-1}}$. Therefore v is not a vertex dead point, a contradiction.

If $\overline{bp}$ crosses ∂P an even number of times and $\overrightarrow{bp} \cdot \overrightarrow{pp_{i-1}} > 0$ and $\overrightarrow{bp} \cdot \overrightarrow{pp_{i+1}} > 0$, then p is constrained to move along either $\overrightarrow{pp_{i-1}}$ or $\overrightarrow{pp_{i+1}}$. Since $bpp_{i-1} > 90°$ and $bpp_{i+1} > 90°$, this moves the point farther from b. Therefore, p is a vertex dead point. □

Corollary 6.2.9. *Each edge of ∂P may contain at most 1 dead point.*

Proof. If d_1 and d_2 are two dead points on an edge e, we have 3 cases:

1. If both are edge dead points, then by Lemma 6.2.7, $\angle bd_1d_2$ and $\angle bd_2d_1$ are both 90°, and since $d_1 \neq d_2$, $\angle d_1bd_2 = \theta > 0$. Therefore, the triangle bd_1d_2 has more than 180°, a contradiction.

2. If both are vertex dead points, then by Lemma 6.2.8, we have $|\overrightarrow{bd_1}| < |\overrightarrow{bd_2}|$ and $|\overrightarrow{bd_2}| < |\overrightarrow{bd_1}|$, a contradiction.

3. If d_1 is a vertex dead point and d_2 is an edge dead point, then by Lemma 6.2.7, and $bd_2d_1 = 90°$ and by Lemma 6.2.8 $\angle bd_1d_2 > 90°$ and so, triangle bd_1d_2 has more than 180°. Again, this is a contradiction.

□

Lemma 6.2.10. *Let P be a simple polygon on n vertices. If $D(b)$ is the set of dead points with respect to b, then $0 \leq |D(b)| \leq n - 3$*

Proof. If P is convex, then there are no dead points, satisfying the lower bound. Since $b \in P$, at least 3 edges of P must have a point visible to b meaning there would necessarily be an odd number of crossing with these edges and a ray centered at b. By Lemma 6.2.6, this implies that these three edges cannot have any dead points. Since no edge can have more than 1 dead point, by corollary 6.2.9, this implies that the upper bound of $n-3$ cannot be exceeded. An example achieving the upper bound is shown in figure 6.7. □

Lemma 6.2.11. *The set of dead regions, $R(b)$, along with the attraction region of b, $A(p)$, partitions the polygon P.*

Proof. We see that every point must eventually either reach b or be forced to stop at a dead point, so these sets cover P. We remove any ambiguity about the movement of a point on a reflex vertex by assuming it always falls to the

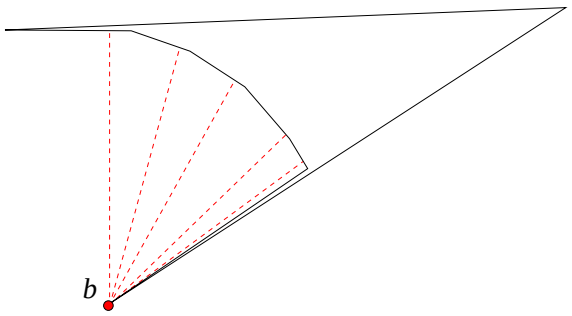


Figure 6.7: A simple polygon P can have at most $n-3$ dead points. In this example, $n=8$, the beacon is at b and the dotted lines indicate perpendiculars with ∂P. Since we have 5 such perpendiculars, $|D(b)|=5$, which agrees with $n-3=8-3=5$.

left of $\overrightarrow{bp}$. Then, every point follows a unique path induced by the beacon b, as the rules for all possible positions are fixed. It follows that a point cannot terminate at two different dead points, d_1 and d_2. □

Definition 6.2.12. *We call a vertex p a* cut-vertex *if it is reflex and the segment from p along $\overrightarrow{bp}$ until it crosses the boundary of P again lies inside the polygon. Every cut-vertex p has associated with it a* ray-vertex *q_p, which is the first point of intersection of the cut-vertex ray with the polygon.*

Combinatorially, there are 4 conceivable types of cut-vertices, p_i, split into 3 classes; ones where p_{i-1} and p_{i+1} lie on opposite sides of line $\overline{bp}$ and the two cases corresponding to p_{i-1} and p_{i+1} lying on the same side of $\overline{bp}$, either left or right, respectively. Given the requirement of parity and that the polygon is oriented counterclockwise, only one direction is valid per class of cut-vertex, and so there are three types of cut-vertices that are possible. Call these class I, II, and III. See Figure 6.8.

Lemma 6.2.13. *1. A vertex is a class I cut-vertex iff it is reflex and $\overrightarrow{bp_i} \times \overrightarrow{p_ip_{i+1}} < 0$ and $\overrightarrow{bp_i} \times \overrightarrow{p_{i-1}p_i} < 0$*

2. A vertex is a class II cut-vertex iff it is reflex and $\overrightarrow{bp_i} \times \overrightarrow{p_ip_{i+1}} > 0$ and $\overrightarrow{bp_i} \times \overrightarrow{p_{i-1}p_i} < 0$

3. A vertex is a class III cut-vertex iff it is reflex and $\overrightarrow{bp_i} \times \overrightarrow{p_ip_{i+1}} < 0$ and $\overrightarrow{bp_i} \times \overrightarrow{p_{i-1}p_i} > 0$

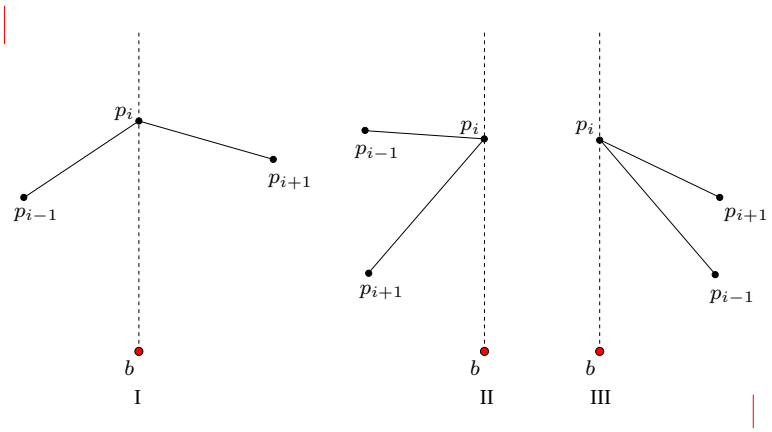


Figure 6.8: There are three classes of cut-vertices.

Proof. From Lemma 6.2.6, we know that in order for the ray $\overrightarrow{bp_i}$ to be contained in the polygon, it must have crossed an even number of edges of P. Since P is oriented counterclockwise, this means that the top edge on either side of $\overrightarrow{bp_i}$ must be ordered from left to right. If we hold $\overrightarrow{bp_i}$ as fixed vertically, this means that the top edge has a negative cross product with $\overrightarrow{bp_i}$. Then, the class definition simply considers all cases given that the vertices are sequentially linked $p_{i-1} \to p_i \to p_{i+1}$ along the counterclockwise orientation of the polygon. □

Definition 6.2.14. *We make a further distinction, and say a cut-vertex is called a split-vertex if:*

1. *It is class I,* $\overrightarrow{bp_i} \cdot \overrightarrow{p_ip_{i+1}} < 0$, *and* $\overrightarrow{bp_i} \cdot \overrightarrow{p_ip_{i-1}} < 0$.
2. *It is class II, and* $\overrightarrow{bp_i} \cdot \overrightarrow{p_ip_{i-1}} < 0$.
3. *It is class III, and* $\overrightarrow{bp_i} \cdot \overrightarrow{p_ip_{i+1}} < 0$.

Definition 6.2.15. *If p_i is a split-vertex, with corresponding ray-vertex q_i, then the line segment $\overline{p_iq_i}$ is called a dead edge.*

Theorem 6.2.16. *If P is a simple polygon and $e = \overline{p_iq_i}$ is a dead edge, then e is the boundary between two dead regions, or the boundary between a dead region and $A(p)$.*

Proof. e partitions P into two pieces, P_L and P_R. The convention mentioned in Lemma 6.2.11 means that the points on e move to the left and so are part of P_L. Since e is parallel to $\overrightarrow{bp_i}$, the direct action of b can never pull a point from P_L to P_R or vice versa. Therefore, the only possible way for a point to move from one side of e to the other is to move unconstrained until reaching ∂P and then slide along an edge. In order to slide along an edge across e, it must pass p_i, and therefore must slide along p_ip_{i+1}, or $p_{i-1}p_i$, depending on whether it started in P_L or P_R. Since p_i is a split-vertex, then regardless what class cut-vertex it is, due to the angles defined for a split-vertex, a point on edge p_ip_{i+1} or $p_{i-1}p_i$ is pulled away from p_i, and can never reach it. Therefore, since the polygon is simple, the points on one side of e cannot terminate in the same location as the points on the right side and so are in different regions.

Furthermore, e is the boundary of at most two regions as the unconstrained points all travel parallel to their ray from b and terminate at the same point, meaning that points on a given ray are in the same region. Therefore, they are in different regions and e is their boundary. □

Theorem 6.2.17. *A boundary edge of a region is either an edge of P, a dead edge, or an edge of the form (p_j, q_i) or (q_i, p_j) for some ray-vertex q_i.*

Proof. All edges of P are boundaries of regions by Lemma 6.2.11. We see that edges of the form (p_j, q_i) or (q_i, p_j) are actually edges of the polygon P that have been split by a ray vertex, which does not change the fact that they bound P and therefore bound some regions partitioning P. We have already seen that all dead edges are boundaries of regions by Theorem 6.2.16.

Take a boundary component c of a region that is not an edge of P, or an edge of the form (p_j, q_i) or (q_i, p_j) for some ray-vertex q_i. If some length of c is not parallel to the ray from b, then the unconstrained attraction from b will pull points across c, implying that the two sides share a dead region. Therefore, c must be a straight segment parallel to the ray from b. Now, c must intersect the polygon at two locations, say s_1 and s_2, with s_1 closer to b. All points on c slide down c to s_1 under the influence of b. If s_1 is on the interior of an edge, there are two cases. If s_1 is a dead point, then points on both sides of the edge of s_1 slide to s_1, implying that c is in the interior of the dead region of s_1, contradiction. If s_1 is not a dead point, then it will slide along the edge, either left or right. In both cases, points from both sides of c end at the same dead point, so c is not on a boundary. Therefore, s_1 is a vertex. We see that the conditions that force all points from one side of c to a

different region than all points on the other side of c are exactly the conditions that make c a split-vertex.

Therefore the boundary edges of regions in the attraction arrangement are exactly the edges of P, dead edges, or edges of the form (p_j, q_i) or (q_i, p_j) for some ray-vertex q_i. □

The previous theorem forms the idea for the attraction region algorithm. First, we find the split-vertices and compute ray-shots to obtain the ray-vertices. Then, the attraction arrangement is exactly a decomposition of P into polygons, each of which contains a single dead point, or in the case of the attraction region (which is also a polygon), b. The arrangement therefore consists of the dead regions and the attraction region.

The Algorithm for Computing the Attraction Region of a Beacon

Here, we first give an algorithm for calculating the attraction region $A(b)$ of a beacon b in a simple polygon P and then consider the case for polygons with holes. Presume the vertices of P are given in counterclockwise order in a linked list, or similar data structure. Iterate through them in order. Initialize the attraction arrangement with the polygon P.

For each reflex vertex $p_i \in P$, check if one of the following holds:

1. $\overrightarrow{bp_i} \times \overrightarrow{p_i p_{i+1}} < 0$ and $\overrightarrow{bp_i} \times \overrightarrow{p_{i-1} p_i} < 0$ and $\overrightarrow{bp_i} \cdot \overrightarrow{p_i p_{i+1}} < 0$ and $\overrightarrow{bp_i} \cdot \overrightarrow{p_i p_{i-1}} < 0$.
2. $\overrightarrow{bp_i} \times \overrightarrow{p_i p_{i+1}} > 0$ and $\overrightarrow{bp_i} \times \overrightarrow{p_{i-1} p_i} < 0$ and $\overrightarrow{bp_i} \cdot \overrightarrow{p_i p_{i-1}} < 0$.
3. $\overrightarrow{bp_i} \times \overrightarrow{p_i p_{i+1}} < 0$ and $\overrightarrow{bp_i} \times \overrightarrow{p_{i-1} p_i} > 0$ and $\overrightarrow{bp_i} \cdot \overrightarrow{p_i p_{i+1}} < 0$.

If any of those three conditions holds, shoot a ray from p_i along $\overrightarrow{bp_i}$ to find q_i and update the attraction arrangement with edge (p_i, q_i). Do this for all $p_i \in P$ and the result is a partitioning of the polygon into $A(b)$ and the set of dead regions $R(b)$. We can obtain $A(b)$ by walking counterclockwise along ∂P, beginning at a visible vertex of b and following dead edges when they are incident to a vertex along the boundary, and stopping when the first (visible) vertex is reached again.

We now consider the run time performance. We can check each vertex to see if it is a split-vertex in constant time, yielding $O(n)$ time to check all the vertices. We can preprocess to shoot rays in $O(n \log n)$ time and then find q_i

in $O(\log n)$ per query [16]. Since there are possibly a linear number of dead points, this yields in total an $O(n \log n)$ runtime.

When P has $h \geq 1$ holes, the algorithm for computing the attraction from a beacon b in P is nearly identical to the simple case. We again compute ray shots from each split-vertex of P. Every non-degenerate hole (having at least 3 vertices) has a split-vertex on it with respect to b. For degenerate holes, we can simply eliminate holes that are points from the input and for a segment (u, v), if b is to the left or on the perpendicular through one endpoint and to the right or on the perpendicular of the other, then we do ray shots through both endpoints. Else, we do a ray shot through the endpoint that is furthest from b. These ray shots connect the entire polygon.

Once all the ray shots have been computed, we begin constructing the boundary of $A(b)$ by starting at a visible vertex of P from b and walking along the boundary of P in the counterclockwise direction (or clockwise direction if we are walking along a hole), and following ray shot edges when they arise. It may be the case that certain edges are encountered more than once. In such instances, we delete the edge from our construction since it is not a boundary edge of $A(b)$.

The run time for this generalized algorithm is dominated by the complexity of computing $O(n)$ ray shots in a polygon with holes. In particular, it takes $O(\sqrt{h}n + h^{\frac{3}{2}} \log h + n \log n)$ time to preprocess and $O(\sqrt{h} \log n)$ to conduct a ray shot [16]. This gives us a total running time of $O(\sqrt{h}n \log n)$.

6.2.3 Routing with a Discrete Set of Candidate Beacons

Using the above attraction region algorithm, we find a *minimum beacon path* between two points s, t, where beacons are chosen from a discrete set of m candidates. A minimum beacon path from s to t is the smallest possible collection of points $b_1, b_2, \ldots, b_k$ in P with the property that b_1 attracts s, b_{i+1} attracts b_i for $i = 1, \ldots, k-1$, and t attracts b_k. The minimum beacon path algorithm constructs a digraph G whose vertices are the candidate locations and which has the edge (u, v) if $u \in A(v)$. The minimum beacon path is then given by the shortest $s - t$ path in G.

Theorem 6.2.18. *A minimum beacon path from s to t, chosen from a set of m candidate locations in a polygon P can be found in time $O(mn \log n + m^2)$ for simple polygons, and $O(\sqrt{h}mn \log n + m^2)$ for polygons with holes.*

6.3 Hardness of Beacon Coverage

We establish the hardness of covering a simple polygon with a minimum cardinality set of becaons. The proof is again based on a reduction from the LINE HITTING problem.

Theorem 6.3.1. *It is NP-hard to find a minimum cardinality set of beacons whose union covers a simple polygon.*

Proof. Given an instance of the LINE HITTING problem, we construct a 'spike box' large enough for its rectangular body to contain a positive length segment of each line and all intersection points formed by the arrangement. The spike box contains an arrow shaped spike gadget for each line in the arrangement that protrudes from the body. We place a spike gadget at either location where a line exits the body of the spike box. Each gadget has an arbitrarily small entrance (see Figure 6.9. The minimum number of beacons to cover the tip of each spike is exactly the number of beacons needed to cover the whole spike box. This is because any beacon placed in the convex body of the spike box covers the body of the spike box. Therefore, if any of the spike-covering beacons are in the body, each spike is covered as well as the body, so the entire polygon is covered. If all the spike-covering beacons are in the spikes, then they are each in the region of points $R(p)$ that attract exactly one tip, say p. Since all intersections of regions $R(p)$ for each p are contained in the body of the spike-box, each beacon is responsible for only one tip, and so we can move it, as long as it remains in $R(p)$. Specifically, we can move it to the body of the spike-box, and cover the entire polygon with the same number of beacons.

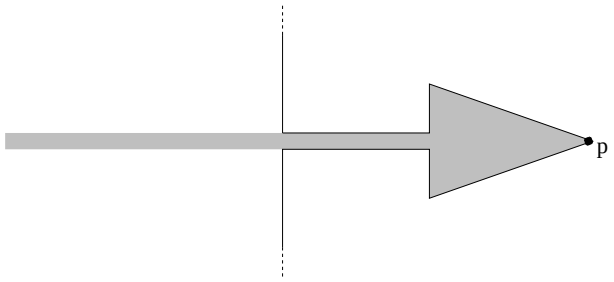


Figure 6.9: An 'arrow spike'. p is attracted by all points in the grey shaded region, which extends into the body of the spike box.

If we can cover each line in the arrangement with k points, then we can cover the polygon with k beacons. Given an instance of k points covering the

lines, place k beacons in their corresponding places in the spike box. Since each line is covered, there is a beacon in $R(p)$ for each spike tip p, and so we can cover the polygon with k beacons. If we can cover the polygon with k beacons, then we can cover the lines with k points. Given the beacon placement, place the points on the corresponding points of the lines. Since each spike-tip is covered, there is a beacon in each $R(p)$ and so there is a point on each line, covering the lines with k points. □

6.4 Conclusion

We have considered beacon-based point-to-point routing and coverage problems motivated by sensor network applications. We showed that $\lfloor \frac{n}{2} \rfloor - 1$ beacons are sometimes necessary and always sufficient to route between any pair of points in a simple polygon P. We developed several algorithms to compute structures to aid in beacon routing. Using these algorithms we can approximate minimum beacon paths for general beacons and find exact paths for discrete beacon sets in polynomial time.

We demonstrated that finding a minimum cardinality set of beacons to route from any source point $s \in P$ to a given destination $t \in P$ or from a particular source point to any destination is NP-hard. We showed that it is NP-hard to find a minimum cardinality set of beacons to cover a simple polygon.

A remaining open question is whether there exists an exact algorithm for computing a minimum beacon path in polynomial time or that finding such a path is NP-hard.

References

[1] M. Abellanas, J. García, G. Hernández-Peñalver, F. Hurtado, O. Serra, and J. Urrutia. Onion polygonizations. *Information Processing Letters*, 57(3):165–173, 1996.

[2] L. Addario-Berry, O. Amini, J.-S. Sereni, and S. Thomass. Guarding art galleries: The extra cost for sculptures is linear. In J. Gudmundsson, editor, *Algorithm Theory SWAT 2008*, volume 5124 of *Lecture Notes in Computer Science*, pages 41–52. Springer Berlin / Heidelberg, 2008.

[3] P. K. Agarwal, F. Hurtado, G. T. Toussaint, and J. Trias. On polyhedra induced by point sets in space. *Discrete Applied Mathematics*, 156(1):42 – 54, 2008.

[4] Y. Amit, J. S. B. Mitchell, and E. Packer. Locating guards for visibility coverage of polygons. *International Jounral of Computational Geometry and Applications*, 20(5):601–630, 2010.

[5] E. Arkin, S. Fekete, F. Hurtado, J. Mitchell, M. Noy, V. Sacristán, and S. Sethia. On the reflexivity of point sets. In B. Aronov, S. Basu, J. Pach, and M. Sharir, editors, *Discrete and Computational Geometry: The Goodman-Pollack Festschrift*, volume 25 of *Algorithms and Combinatorics*, pages 139–156. Springer-Verlag, 2003.

[6] B. Ben-Moshe, M. J. Katz, and J. S. B. Mitchell. A constant-factor approximation algorithm for optimal terrain guarding. In *Proc. of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 515–524, Vancouver, British Columbia, 2005. Society for Industrial and Applied Mathematics.

[7] P. Berman and M. Karpinski. On some tighter inapproximaability results. In *Proc. Intl. Colloquium on Automata, Languages and Programming (ICALP'99)*, pages 200–209, 1999.

[8] T. Biedl, M. T. Irfan, J. Iwerks, J. Kim, and J. S. B. Mitchell. Guarding polyominoes. In *Proc. of the 27th Annual Symposium on Computational Geometry (SoCG 2011)*, pages 387–396, Paris, France, 2011. Association for Computing Machinery.

[9] M. Biro, J. Gao, J. Iwerks, I. Kostitsyna, and J. S. B. Mitchell. Beacon based routing and coverage. In *Proc. of the 21st Fall Workshop on Computational Geometry*, pages 16–17, New York, NY, November 2011.

[10] M. Biro, J. Gao, J. Iwerks, I. Kostitsyna, and J. S. B. Mitchell. Beacon based structures in polygonal domains. In *Abstracts of the 1st Computational Geometry: Young Researchers Forum*, Chapel Hill, NC, June 2012.

[11] I. Bjorling-Sachs and D. Souvaine. An efficient algorithm for placing guards in polygons with holes. *Discrete and Computational Geometry*, 13:77–109, 1995.

[12] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[13] B. Brodén, M. Hammar, and B. J. Nilsson. Guarding lines and 2-Link polygons is APX-Hard. In *Proc. of the 13th Canadian Conference on Computational Geometry*, pages 45–48, 2001.

[14] B. Chazelle. *Computational geometry and convexity*. PhD thesis, Yale University, 1980.

[15] B. Chazelle. On the convex layers of a planar set. *IEEE Transactions on Information Theory*, 31:509–517, 1985.

[16] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

[17] V. Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory Series B*, 18:39–41, 1975.

[18] L. Deneen and G. Shute. Polygonizations of point sets in the plane. *Discrete and Computational Geometry*, 3:77–87, 1988.

[19] A. Efrat and S. Har-Peled. Guarding galleries and terrains. *Information Processing Letters*, 100(6):238–245, Dec. 2006.

[20] Q. Fang, J. Gao, L. Guibas, V. de Silva, and L. Zhang. GLIDER: Gradient landmark-based distributed routing for sensor networks. In *Proc. of the 24th Conference of the IEEE Communication Society (INFOCOM)*, volume 1, pages 339–350, March 2005.

[21] S. P. Fekete. On simple polygonizations with optimal area. *Discrete and Computational Geometry*, 23:73–110, 2000.

[22] R. Fonesca, S. Ratnasamy, J. Zhao, C. T. Ee, D. Culler, S. Shenker, and I. Stoica. Beacon vector routing: Scalable point-to-point routing in wireless sensornets. In *Proc. of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, pages 329–342, May 2005.

[23] S. Ghosh. Approximation algorithms for art gallery problems. In *Proc. of Canadian Information Processing Society Congress*, pages 429–434, 1987.

[24] S. Ghosh. Approximation algorithms for art gallery problems in polygons. *Discrete Applied Mathematics*, 158(6):718–722, 2010.

[25] R. L. Graham. An efficient algorithm for determining the convex hull of finite planar set. *Information Processing Letters*, 1:132–133, 1972.

[26] B. Grünbaum. Hamiltonian polygons and polyhedra. *Geombinatorics*, 3:83–89, 1994.

[27] E. Györi. A short proof of the rectilinear art gallery theorem. *SIAM Journal on Algebraic and Discrete Methods*, 7(3):452–454, 1986.

[28] J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. *BIT Numerical Mathematics*, 32:249–267, 1992.

[29] F. Hoffman. On the rectilinear art gallery problem. In *Proc. of the 17th International Colloquium on Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 717–728. Springer, 1990.

[30] J. Iwerks. Guarding polyforms. In *Abstracts of the 1st Computational Geometry: Young Researchers Forum*, Chapel Hill, NC, June 2012.

[31] J. Iwerks and J. S. B. Mitchell. Spiral serpentine polygonization of a planar point set. In *Proc. of XIV Spanish Meeting on Computational Geometry*, pages 181–184, Alcalá de Henares, Spain, 2011. Centre de Recerca Matemática.

[32] J. Iwerks and J. S. B. Mitchell. The art gallery theorem for simple polygons in terms of the number of reflex and convex vertices. *Information Processing Letters (to appear)*, 2012.

[33] J. Iwerks and J. S. B. Mitchell. Spiral serpentine polygonization of a planar point set. *Lecture Notes in Computer Science (to appear)*, 2012.

[34] J. Kahn, M. Klawe, and D. Kleitman. Traditional galleries require fewer watchmen. *SIAM Journal on Algebraic and Discrete Methods*, 4(2):194–206, June 1983.

[35] B. Karp and H. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 243–254, 2000.

[36] M. J. Katz and G. S. Roisman. On guarding the vertices of rectilinear domains. *Computational Geometry: Theory and Applications*, 39(3):219–228, 2008.

[37] D. Lee and A. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32(2):276–282, 1986.

[38] M. Newborn and W. O. J. Moser. Optimal crossing-free hamiltonian circuit drawings of K_n. *Journal of Combinatorial Theory, Series B*, 29(1):13 – 26, 1980.

[39] A. Nguyen, N. Milosavljevic, Q. Fang, J. Gao, and L. J. Guibas. Landmark selection and greedy landmark-descent routing for sensor networks. In *Proc. of the 26th Conference of the IEEE Communications Society (INFOCOM'07)*, May 2007.

[40] B. J. Nilsson. Approximate guarding of monotone and rectilinear polygons. In *Automata, Languages and Programming*, pages 1362–1373. Springer, 2005.

[41] J. O'Rourke. An alternate proof of the rectilinear art gallery theorem. *Journal of Geometry*, 21(1):118–130, Dec. 1983.

[42] J. O'Rourke. *Art gallery theorems and algorithms*. Oxford University Press, Inc., 1987.

[43] V. Pinciu. Pixel guards in polyominoes. In *Proc. of the Cologne-Twente Workshop on Graphs and Combinatorial Optimizations (CTW 2010)*, pages 137–140, 2010.

[44] L. V. Quintas and F. Supnick. On some properties of shortest hamiltonian circuits. *American Mathematical Monthly*, 72:977–980, 1965.

[45] D. Schuchardt and H. Hecker. Two NP-Hard Art-Gallery problems for Ortho-Polygons. *Mathematical Logic Quarterly*, 41(2):261–267, 1995.

[46] M. I. Shamos and D. Hoey. Closest-point problems. In *Proc. of the 16th IEEE Symposium on Foundations of Computer Science*, pages 151–162, Berkeley, CA, 1975.

[47] T. Shermer. Recent results in art galleries [geometry]. *Proc. of the IEEE*, 80(9):1384–1399, 1992.

[48] H. Steinhaus. *One Hundred Problems in Elementary Mathematics*. Dover Publications, Inc., 1964.

[49] J. Urrutia. Art gallery and illumination problems. *Handbook of Computational Geometry*, pages 973–1027, 2000.

[50] C. Worman and J. M. Keil. Polygon decomposition and the orthogonal art gallery problem. *International Journal of Computational Geometry and Applications*, 17(2):105–138, 2007.

Appendix A

Here we prove that $\lfloor \frac{3m}{11} \rfloor + 1$ pixel guards (we will just call these 'guards' in this section) are sufficient to cover a polyomino on m pixels. We briefly introduce some notation. Presume that $a_1 - a_2 - a_3$ are three pixels where a_1 and a_3 are different, but both adjacent to a_2. We say that $a_1 - a_2 - a_3$ is a *corner* if a_1 and a_3 share a point, and *straight* otherwise. If a_1 is the parent of a_2 in the BFS tree, then we also call a_3 a *corner child* of a_2 or the *straight child* of a_2 if $a_1 - a_2 - a_3$ is a corner or straight, respectively.

The proof of the bound will almost exclusively work on the dual graph of the polyomino, and specifically, on the BFS tree T defined earier. As before, we use the same notation for a pixel and for the node in the BFS tree that represents the pixel. We first give two lemmas that are simple observations about visibility, phrased in terms of the dual graph. Subsequent lemmas will be nested in the body of Theorem 3.2.8's proof.

Lemma A.0.1. *If v and w are two pixels that have distance ≤ 2 in the dual graph, then a guard at v will cover all of w.*

Proof. If v and w are adjacent, then the claim is trivial. Let x be the common neighbor of v and w. Then x contains points of the guard at v, and $x \cup w$ forms a rectangle, hence a convex shape. So the points in $x \cap v$ cover all of $x \cup w$. □

Lemma A.0.2. *If $v_1 - v_2 - v_3 - v_4$ is a path in the dual graph, and $v_1 - v_2 - v_3$ is a corner, then a guard at v_1 will cover v_4.*

Proof. Since $v_1 - v_2 - v_3$ is a corner, v_1 and v_3 have one point in common. This point, which belongs to the guard at v_1, covers the convex shape $v_3 \cup v_4$. □

Proof of Theorem 3.2.8. We prove this by induction on m. In the base case, there exists a node that has distance ≤ 6 to all other nodes. We will deal with this after the induction step.

Assume for the induction step that for any node r some other node has distance ≥ 7 from r. We compute a BFS tree in the dual graph, using the special preference search rules that were previously described in Section 3.2.2. Let T be the BFS tree, and for any node v in T let T_v be the subtree rooted at v, i.e., v and all its descendants. As we will show below, some rooted subtree T_v has k nodes and can be covered with $\lfloor 3k/11 \rfloor$ guards. We call such a subtree *pixel-good*. We can remove T_v from P and the remainder is still connected (since $T - T_v$ is). By induction, $P - T_v$ can be covered with $\lfloor (3m-k)/11 \rfloor + 1$ guards, which proves the claim.

Let q be a lowest leaf of T, let p_1 be its parent, p_2 be its grandparent, p_3 be its great-grandparent, and generally p_i be its great-great-...-great-grandparent, where "great" is repeated $i-2$ times. We know that p_7 exists, since some node has distance at least 7 from r, and hence the BFS tree, which measures this distance, has height at least 7. Through an extensive case analysis, we will now show that some rooted subtree of T_{p_6} is pixel-good. We start at T_{p_2}, and then work our way up the generations as needed.

T_{p_2} (and other subtrees of height ≤ 2): Since T_{p_2} has height 2, one guard at p_2 covers all of T_{p_2} by Claim A.0.1. So if $|T_{p_2}| \geq 4$, then it has a guard-pixel ratio of $\frac{1}{4} < \frac{3}{11}$ and it is pixel-good. We state this simple observation separately, and generalized to subtrees of any height ≤ 2, because it will be useful later.

Lemma A.0.3. *Let v be a node such that T_v has height ≤ 2. Then T_v is pixel-good unless $|T_v| = 0, 1, 2, 3$.*

Note that T_{p_2} always has height 2 and size at least 3, so by Lemma A.0.3 we are done unless T_{p_2} has size 3, i.e., it is a 2-path.

T_{p_3} (and other subtrees of height ≤ 3): If T_{p_2} has size 3, then we go up a generation and study T_{p_3}. Let p_2' and p_2'' be the siblings of p_2, which may or may not exist. Let $T_{p_2'}$ and $T_{p_2''}$ be empty subtrees if they do not exist. By Lemma A.0.3, and after possible renaming, we can assume that $|T_{p_2''}| \leq |T_{p_2''}| \leq 3$, otherwise there is a pixel-good polyomino in one of these subtrees and we are done. We distinguish cases that are illustrated in Figure A.1.

(3a) $T_{p_2'}$ and $T_{p_2''}$ have size ≤ 1. Recall that T_{p_2} can be covered with one guard at p_2. Such a guard also covers p_3 and all of $T_{p_2'}$ and $T_{p_2''}$, since they have size ≤ 1. So T_{p_3}, which has at least 4 pixels, can be covered with 1 guard and is pixel-good.

(3b) The sizes of $T_{p_2}, T_{p_2'}, T_{p_2''}$ are $(3, 2, 0)$, and p_3 has a sibling. Since p_3 has a sibling, p_2 and p_2' cannot be on opposite sides of p_3 by Lemma 3.2.5. Hence a guard at p_2 will also cover the child of p_2' by Lemma A.0.2. So T_{p_3}, which has 6 pixels, can be covered with one guard and is pixel-good.

(3c) The sizes of $T_{p_2}, T_{p_2'}, T_{p_2''}$ are $(3, 2, 0)$, and p_3 has no sibling. In this case, T_{p_3} forms a 5-path. This case can sometimes not be resolved at this level, and we will deal with it later.

(3d) The sizes of $T_{p_2}, T_{p_2'}, T_{p_2''}$ are $(3, 2, 1)$ or $(3, 2, 2)$. In this case, p_3 has three children and hence no sibling by Lemma 3.2.5, which means that T_{p_4} has one more pixel than T_{p_3}. Hence, T_{p_4} has 8 pixels and can be covered with two guards at p_2 and p_3, so it is pixel-good.

(3e) The sizes of $T_{p_2}, T_{p_2'}, T_{p_2''}$ are $(3, 3, x)$, for some $x \in \{0, 1, 2\}$, and p_3 has no sibling. Place guards at p_2 and p_2'. If p_2'' exists, then it must form a corner with at least one of these guards, so this covers $T_{p_2''}$ as well by Lemma A.0.2. Hence, T_{p_4} can be covered with 2 guards, and it has at least 8 pixels, so it is pixel-good.

(3f) The sizes of $T_{p_2}, T_{p_2'}, T_{p_2''}$ are $(3, 3, x)$, for some $x \in \{0, 1, 2\}$, p_3 has a sibling, and $T_{p_2'}$ has height 1. Since p_3 has a sibling, by Lemma 3.2.5, p_3 cannot have two children on opposite sides and so actually $x = 0$ and p_2 and p_2' are not on opposite sides of p_3. By Lemma A.0.2 a guard at p_2 then covers all of $T_{p_2'}$ as well as all of T_{p_2}. So T_{p_3} can be covered with one guard and is pixel-good.

(3g) The sizes of $T_{p_2}, T_{p_2'}, T_{p_2''}$ are $(3, 3, x)$, for some $x \in \{0, 1, 2\}$, p_3 has a sibling, and $T_{p_2'}$ has height 2. As in the previous case, one argues that $x = 0$. So in this case T_{p_3} is a 6-path and $p_2 - p_3 - p_2'$ is a corner. This is another special case that we will deal with later.

(3h) The sizes of $T_{p_2}, T_{p_2'}, T_{p_2''}$ are $(3, 3, 3)$. Since p_3 has three children, it has no siblings, so $|T_{p_4}| = 11$. We can cover p_4 with three guards at p_2, p_2' and p_2'', and hence T_{p_4} is pixel-good.[1]

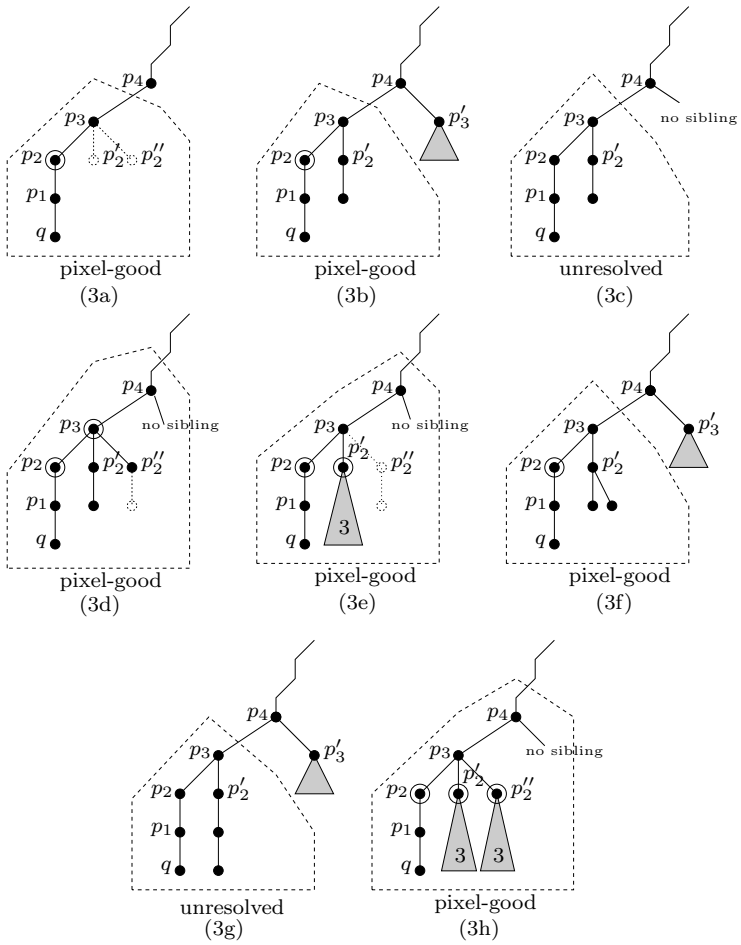


Figure A.1: The cases for a subtree of height 3. Dashed nodes need not exist. Circled nodes are used as guards.

(6′) **and** (7′) **and how to cover them:** We have found a pixel-good subtree in all but two cases:

- T_{p_3} is a 5-path with p_3 one of its median nodes, and p_3 has no siblings, or
- T_{p_3} is a 6-path with p_3 the median node, and p_3 has siblings.

To deal with these special cases, we will go up one more generation to T_{p_4}

[1]This is one case where the bound is tight; this is also the case that occurs in our lower-bound example of Figure 3.4.

and sometimes T_{p_5}. Before doing this, we study further the geometric structure of T_{p_3}, and how to cover it.

Definition A.0.4. *Let* $(6')$ *be the class of polyominoes for which the dual graph is a 5-path* $v_1 - \cdots - v_6$*, and* $v_2 - v_3 - v_4 - v_5$ *are straight.* v_3 *and* v_4 *are called the* median-nodes *of such a polyomino.*

Let $(7')$ *be the class of polyominoes for which the dual graph is a 6-path* $v_1 - \cdots - v_7$*.* v_4 *is called the* median-node *of such a polyomino.*

Lemma A.0.5. *Any 6-polyomino* P *that has as its dual a 5-path* $v_1 - \cdots - v_6$ *and is not in* $(6')$ *can be covered with one guard.*

Proof. If v_3 does not cover all of P, then $v_3 - v_4 - v_5$ must be straight by Lemma A.0.2. If v_4 does not cover all of P, then $v_4 - v_3 - v_2$ must be straight by Lemma A.0.2. Therefore, if P cannot be covered with one guard, then $v_2 - v_3 - v_4 - v_5$ is straight and it follows that P is in $(6')$. □

Combining this lemma with the cases where we weren't done yet tells us exactly when considering T_{p_3} (or some other subtree of height ≤ 3) is not enough.

Lemma A.0.6. *Let* v *be a node such that* T_v *has height* ≤ 3 *and* v *has parent* u*. Then some subtree of* T_u *is pixel-good unless*

- $|T_v| = 0, 1, 2, 3$*, or*
- T_v *is in* $(6')$ *with* v *one of its median nodes, and* v *has no siblings, or*
- T_v *is in* $(7')$ *with* v *the median node, and* v *has siblings.*

An instance of $(6')$ or $(7')$ cannot always be covered with one guard, but we can always find a cover with two guards such that these guards also cover many nearby pixels. This will be the crucial insight to cover the (many!) remaining cases. We hence state this as two separate lemmas first, and then mostly only use the lemmas in the later proofs.

Lemma A.0.7. *Let* v *be a node such that* T_v *is in* $(7')$ *with* v *as the median node. Then* T_v *can be covered with two pixel-guards such that also all nodes of distance* ≤ 2 *from* v *are covered.*

Proof. Place two guards at the two children of v; this covers all of T_v. At least one of those children is a corner-child of v, and by Lemma A.0.2 hence covers the parent of v and all its neighbors. □

Lemma A.0.8. *Let v be a node such that has only one child c, and c is median node of a (6′). Then T_v can be covered with two pixel-guards such that also all nodes of distance ≤ 2 of v are covered.*

Proof. Place a guard at v and another at the other median node of the (6′). This covers all nodes of distance ≤ 2 from v by Lemma A.0.1. Also, all neighbors of c either have a pixel or form a corner with a neighbor of c that has a pixel; by Lemma A.0.2 this covers T_v. □

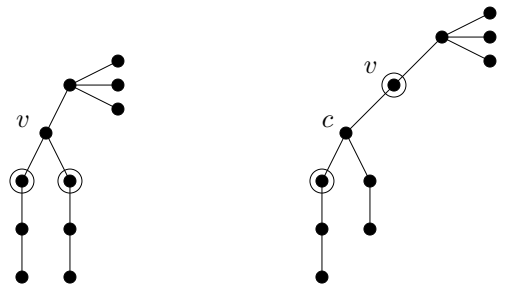


Figure A.2: The special cover for subtrees of size 7. (a) T_v is in (7′). (b) v has a single child whose subtree is in (6′).

T_{p_4} (and other subtrees of height ≤ 4): Now we work on the remaining cases. We left off in the situation where T_{p_3} was either in (6′) or in (7′), and if it was in (6′), then p_3 has no sibling. In all but the last two cases to come below, p_3 has a sibling, and so T_{p_3} is in (7′). We will always use the covering from Lemma A.0.7 for such a T_{p_3}.

Now go up a generation and consider T_{p_4}. Let the siblings of p_3 be p_3' and p_3''. As before, let $T_{p_3'}$ and $T_{p_3''}$ be the (possibly empty) subtrees of the siblings.

We are done if there is a pixel-good subtree in $T_{p_3'}$ or $T_{p_3''}$, so by Lemma A.0.6, we can assume that $T_{p_3'}$ and $T_{p_3''}$ have size 0, 1, 2, 3, 6 or 7. Moreover, if $T_{p_3'}$ has size 7, then it is in (7′) with p_3' as the median. If $T_{p_3'}$ has size 6, then it is in (6′) with p_3' as a median and p_3' has no sibling. But p_3' *has* a sibling, so actually $T_{p_3'}$ cannot have size 6. Therefore $T_{p_3'}$ (and by the same argument $T_{p_3''}$) has size 0, 1, 2, 3, or it is in (7′) with p_3' as the median.

As before we distinguish cases, which are illustrated in Figure A.3. Some of these cases could actually proved more easily (or even omitted altogether) by arguing about the polyomino directly, but as before we will (mostly) argue about the graph to make it possible to "re-use" the proofs for later cases.

(4a) The sizes of $T_{p'_3}, T_{p''_3}$ are $(7, 7)$. Hence T_{p_3}, $T_{p'_3}$ and $T_{p''_3}$ are all in $(7')$ and we cover them with 2 guards each as in Lemma A.0.7. This also covers p_4. So we have used 6 guards for 22 pixels and T_{p_4} is pixel-good.[2]

(4b) The sizes of $T_{p'_3}, T_{p''_3}$ are $(7, 3)$. Cover T_{p_3} and $T_{p'_3}$ with 2 guards each as in Lemma A.0.7 and $T_{p''_3}$ with 1 guard at p''_3.[3] Since p_4 has three children it has no sibling. Hence, T_{p_5} has 19 pixels and can be covered with 5 guards. Since $\frac{5}{19} < \frac{3}{11}$, T_{p_5} is pixel-good.

(4c) The sizes of $T_{p'_3}, T_{p''_3}$ are $(7, 2)$. Cover T_{p_3} and $T_{p'_3}$ with 2 guards each as in Lemma A.0.7. This covers p_4 and p''_3, and studying the geometry of the polyomino, we can now argue that this also coveres the (unique) child of p''_3.

Observe that p_4 has 5 grandchildren, and none of them can be adjacent to p_5 by BFS properties, leaving 5 possible locations. See Figure A.3(4c). Since p_6 exists, the corner-children of p_4 are visited first in the BFS traversal, and so they reach all but one of the grandchildren. So p_3 and p'_3 must be corner-children of p_4, while p''_3 is the straight child of p_4. The child of p''_3 then shares a corner with two of the pixel-guards and is covered. So T_{p_4} has 16 pixels and can be covered with 4 guards. Since $\frac{4}{16} < \frac{3}{11}$, T_{p_4} is pixel-good.[4]

(4d) The sizes of $T_{p''_3}$ are $(7, 1)$ or $(7, 0)$. We cover T_{p_3} and $T_{p'_3}$ with two guards each as in Lemma A.0.7. This also covers p_4 and p''_3 and so T_{p_4} (which has 16 pixels) is pixel-good.

(4e) The sizes of $T_{p'_3}, T_{p''_3}$ are $(3, 3)$. Cover T_{p_3} with two guards as in Lemma A.0.7. Cover $T_{p'_3}$ and $T_{p''_3}$ with 1 guard each at p'_3 and p''_3. Since p_4 has three children, it has no sibling and so T_{p_5} has 15 pixels and can be covered with 4 guards. Since $\frac{4}{15} < \frac{3}{11}$, T_{p_5} is pixel-good.

(4f) The sizes of $T_{p'_3}, T_{p''_3}$ are $(3, 2)$, and $T_{p_4} - T_{p_3}$ (which has size 6) is not in $(6')$. Cover T_{p_3} with two guards as in Lemma A.0.7. Tree $T_{p_4} - T_{p_3}$ can

[2]This case can actually never occur as a polyomino.

[3]One can actually show that the guard at p''_3 is not needed, similarly as for case (4c).

[4]These guards even cover p_5 which has no other children, so itcould be included.

be covered with one guard (Lemma A.0.5), so T_{p_4} can be covered with 3 guards for 13 pixels and T_{p_4} is pixel-good.

(4g) The sizes of $T_{p'_3}, T_{p''_3}$ are $(3,2)$, and $T_{p_4} - T_{p_3}$ is in $(6')$. We will show that this case cannot happen. For if $T_{p_4} - T_{p_3}$ is in $(6')$, then $p'_3 - p_4 - p''_3$ is straight. Therefore $p_5 - p_4 - p_3$ is straight as well and so p_3 is the straight child of p_4. By the BFS property, p'_3 and p''_3 were visited before p_3.

There are only three possible locations for p_2 and its sibling (shown dashed in Figure A.3(4g)). Hence, one of these pixels is adjacent to one of $\{p'_3, p''_3\}$, and would have been made a child of that node, not p_3: a contradiction.

(4h) The sizes of $T_{p'_3}, T_{p''_3}$ are $(3,1)$, $(3,0)$, $(2,2)$, or $(2,1)$. Then $T_{p_4} - T_{p_3}$ has size 4 or 5. So $T_{p_4} - T_{p_3}$ can be covered with one guard, and T_{p_3} can be covered with two guards as in Lemma A.0.7. Therefore T_{p_4}, which has 11 or 12 pixels, can be covered with 3 guards and is pixel-good.

(4i) The sizes of $T_{p'_3}, T_{p''_3}$ are $(2,0)$ and p_4 has no sibling. Then T_{p_5} has 11 pixels and can be covered with 3 guards (two in T_{p_3} by Lemma A.0.7 and one at p_4), so it is pixel-good.

(4j) The sizes of $T_{p'_3}, T_{p''_3}$ are $(2,0)$ and p_4 has a sibling. Cover T_{p_3} with two guards at its children p_2 and p'_2. Similar as in case (4c), we can argue by studying the geometry that this also covers the child of p'_3, and hence T_{p_4} is pixel-good.

After possible exchange of p_2 and p'_2, assume that p_2 is a corner-child of p_3, so $p_2 - p_3 - p_4$ is a corner. Since p_3 has a sibling, by Lemma 3.2.5 $p_2 - p_3 - p'_2$ is a corner. Applying again Lemma 3.2.5, also $p_3 - p_4 - p'_3$ is a corner since p_4 has a sibling. This leaves only two possible locations for p'_3. In one of them p'_3 is adjacent to p_2 and so the guard at p_2 covers p'_3 and its child by Lemma A.0.1.

In the other case (where p'_3 is right of p_4 in Figure A.3(4j)), $p_5 - p_4 - p'_3$ must be a corner (because p_5 must not be adjacent to p_2), which leaves only two positions for the child of p''_3 (which also must not be adjacent to p_5.) Both of these pixels are covered by p_2. Therefore, T_{p_4} has 10 pixels and can be covered with 2 guards, making it pixel-good.

(4k) The sizes of $T_{p'_3}, T_{p''_3}$ are $(1,x)$, with $x \in \{0,1\}$. Cover T_{p_3} with two guards as in Lemma A.0.7; this also covers p_4, p'_3 and p''_3. Hence we can cover T_{p_4}, which has 9 or 10 pixels, with 2 guards, and it is pixel-good.

(4l) The sizes of $T_{p_3'}, T_{p_3''}$ are $(0,0)$ and T_{p_3} is in $(7')$. Cover T_{p_3} with two guards as in Lemma A.0.7; this also covers p_4. Hence we can cover T_{p_4}, which as 8 pixels, with 2 guards, and it is pixel-good.[5]

(4m) The sizes of $T_{p_3'}, T_{p_3''}$ are $(0,0)$ and T_{p_3} is in $(6')$. Hence, p_3 has no siblings and T_{p_3} is in $(6')$ (not in $(7')$ as in all previous cases). This is the same case as (3c), and the only case that we can't resolve at this generation-level.

We summarize the result for T_{p_4}, and generally any node that has a subtree of height ≤ 4.

Lemma A.0.9. *Let v be a node such that T_v has height ≤ 4 and v has parent u. Then some subtree of T_u is pixel-good unless*

- $|T_v| = 0, 1, 2, 3$, *or*
- *v has a single child w, and T_w is in $(6')$ with w one of its median nodes.*

The final case and T_{p_5}: We resolve the one remaining case (4m) by going up yet one more generation. We know that T_{p_4} has size 7 and p_4 has one child that is a median of a $(6')$, so Lemma A.0.8 applies with $v = p_4$.

Consider p_5 and its children p_4, p_4' and p_4'' (not all of which must exist.) We are done if there is a pixel-good subtree in $T_{p_4'}$ or $T_{p_4''}$, so by Lemma A.0.9, we can assume that $T_{p_4'}$ and $T_{p_4''}$ have size 0, 1, 2, 3, or 7 and if they have size 7, they have a single child that is a median of an instance of $(6')$, which means that Lemma A.0.8 applies to them.

We will always use Lemma A.0.8 to cover T_{p_4}. This covers many nearby nodes in the tree, but we have to go through all cases to verify that no extra guards are needed, or that there are enough pixels in them to allow for extra guards. Luckily enough, most of the proofs from cases (4a-m) can be "re-used". More precisely, with the exception of (4c), (4g), (4j) and (4m), we did not use geometry, and we did not use the exact structure of T_{p_3} (and other subtrees of size 7): we only used that T_{p_3} can be covered with 2 guards by Lemma A.0.7.

In the same way, we can prove most cases now by covering T_{p_4} (and other subtrees of size 7) with 2 guards as in Lemma A.0.8. We demonstrate this for (5a) in detail and for the other cases only sketch the idea. Finally we cover all the cases where the proof from p_3 cannot be transferred.

[5]This case was already covered in (3e).

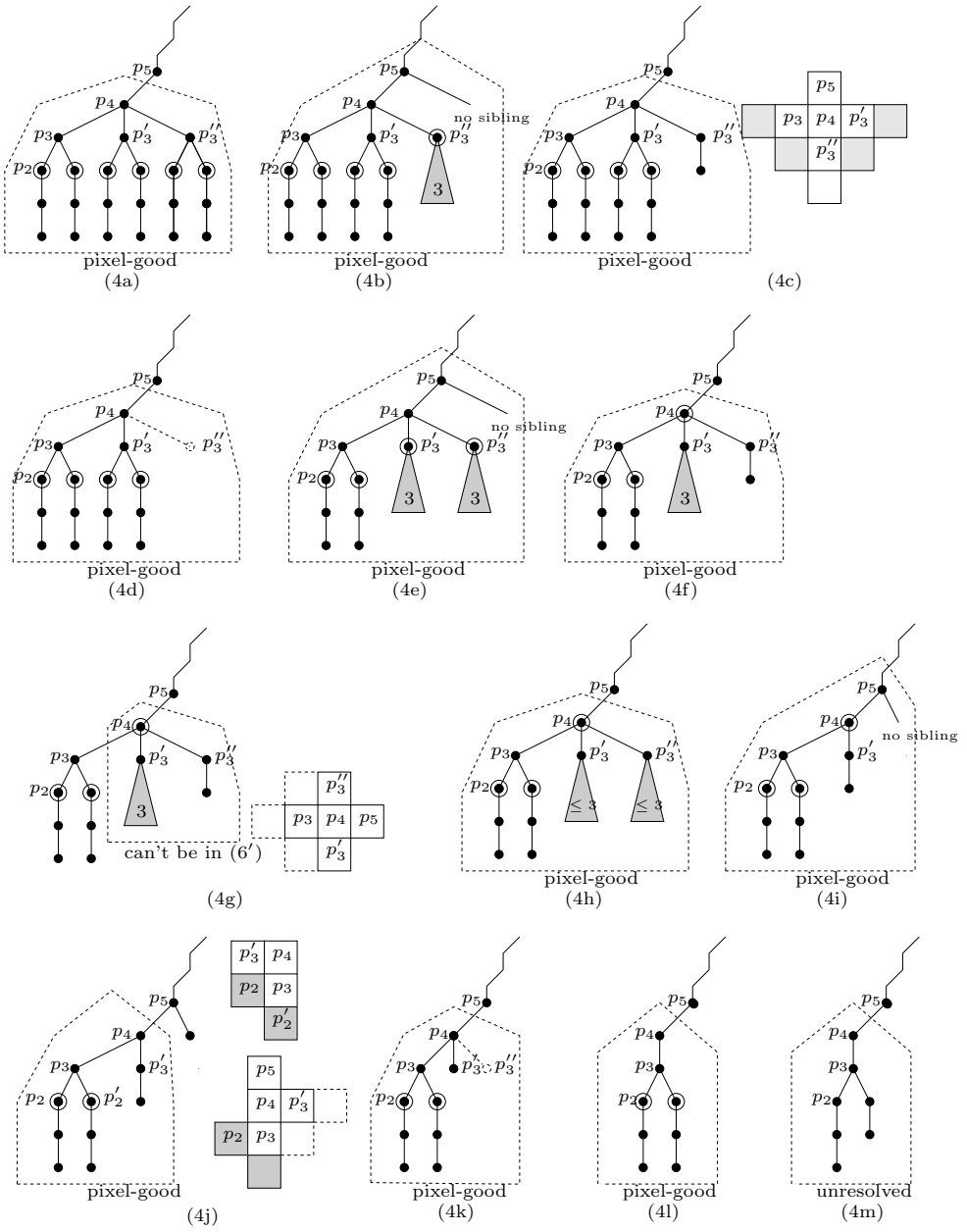


Figure A.3: The cases for a subtree of height 4. Dashed nodes need not exist. Circled nodes are used as guards.

(5a) The sizes of $T_{p'_4}, T_{p''_4}$ are $(7, 7)$. Hence T_{p_4}, $T_{p'_4}$ and $T_{p''_4}$ can be covered with 2 guards each as in Lemma A.0.8. This also covers p_5. So we have

used 6 guards for 22 pixels and T_{p_5} is pixel-good.

(5X) where $X \in \{b, d, e, f, h, i, k, l\}$: The situation is exactly as for case (4X), except that all indices are increased by 1.

Take the proof of case (4X) verbatim, except replace "Lemma A.0.7" by "Lemma A.0.8" and add 1 to all indices. This finds a pixel-good subtree of T_{p_6}.

(5c) The sizes of $T_{p'_4}, T_{p''_4}$ are $(7, 2)$. Cover T_{p_4} and $T_{p'_4}$ as in Lemma A.0.8. This uses p_4 and p'_4 as guards. One of these pixels forms a corner with p''_4, so by Lemma A.0.2 this also covers the child of p''_4.

(5g) The sizes of $T_{p'_4}, T_{p''_4}$ are $(3, 2)$, and $T_{p_5} - T_{p_4}$ is in $(6')$. Cover T_{p_4} as in Lemma A.0.8 with 2 guards, one of which is at p_4. Place a third guard at p'_4. As in the previous case, one of these guards forms a corner with p''_4, so this also covers the child of p''_4 by Lemma A.0.2.

(5j) The sizes of $T_{p'_4}, T_{p''_4}$ are $(2, 0)$. We have two sub-cases

(5j1) p_5 has no sibling. We can then cover T_{p_6} with 3 guards (at p_2, p_4, p_5). Since T_{p_6} has 11 pixels, it is pixel-good.

(5j2) p_5 has a sibling. By Lemma A.0.8, T_{p_4} can be covered with guards at p_2 and p_4. By Lemma 3.2.5 $p_4 - p_5 - p'_4$ is a corner, so this also covers p'_4 and its child. Thus, T_{p_5} can be covered with 2 guards and is pixel-good.

(5l) The sizes of $T_{p'_4}, T_{p''_4}$ are $(0, 0)$. We can then cover T_{p_5}, which has 8 pixels, with two guards at p_2 and p_4, making it pixel-good.

This concludes the inductive step. As for the base case, if we have a polyomino where T has height ≤ 6, then it has $O(1)$ pixels and hence can be covered with $O(1)$ guards. So a bound of $\lfloor 3m/11 \rfloor + O(1)$ is trivial. This can be refined to $\lfloor 3m/11 \rfloor + 1$ by observing that a sufficiently deep tree was required only for two occasions: to be able to go up a generation when needed, and to use Lemma 3.2.5, which requires the existence of a grandparents. Going through cases, one can see that in all of them one extra guard at the root will then suffice to cover the polyomino, so at most $\lfloor 3m/11 \rfloor + 1$ guards are required to cover the polyomino. □

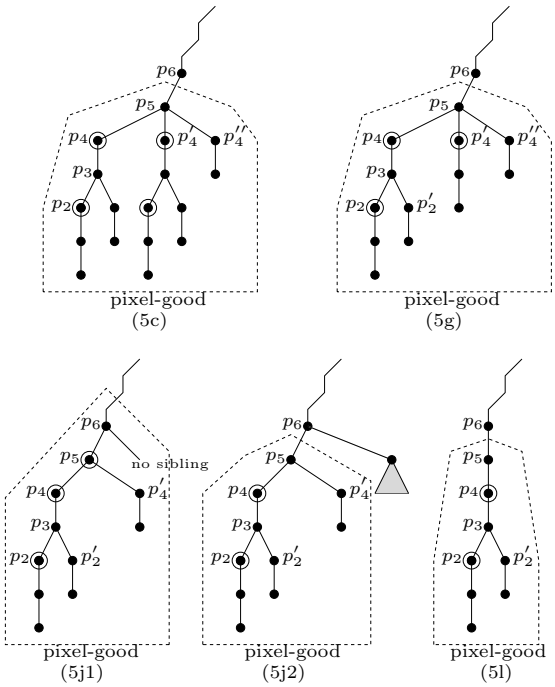


Figure A.4: The cases for a subtree of height 5.