Impossibility of the Dissection of a Cube

Thomas Holme Surlykke

March 19, 2025

Abstract

This entry formalizes Littlewood's argument [2], demonstrating that a 3-dimensional cube cannot be dissected into a finite collection of smaller cubes, each of a different size. The formalization addresses theorem #82, "Dissection of Cubes (J.E. Littlewood's 'elegant' proof)" from Freek Wiedijk's "100 Mathematical Theorems" list [1], and is based upon a prior formalization in Lean [3].

Contents

theory *Cube-Dissection*
imports *Complex-Main HOL−Library.Disjoint-Sets HOL−Library.Infinite-Set*
begin

Proof that a cube can't be dissected into a finite number of subcubes of different size This formalization is heavily inspired by the Lean proof of the same fact, [3]. One goal of this project, is that by restricting to cubes of dimension 3 the logic will be easier to follow

1 Basic definitions

1.1 point and cube definitions

record *point* = *px*:: *real py*::*real pz*::*real*
record *cube* = *point*:: *point width*::*real*
datatype *axis* = *x* | *y* | *z*
abbreviation *coordinate* ≡ *case-axis px py pz*

abbreviation *is-valid* :: *cube* ⇒ *bool* **where** *is-valid c* ≡ (*width c* > *0*)

Min value of cube along given axis

hide-const (**open**) *min*
abbreviation *min* :: *axis* ⇒ *cube* ⇒ *real* **where** *min ax c* ≡ *coordinate ax* (*point c*)

Max value (supremum) along given axis

hide-const (**open**) *max*
abbreviation *max* :: *axis* ⇒ *cube* ⇒ *real* **where** *max ax c* ≡ *min ax c* + *width c*

Sides of a cube. Half-open intervals, so that a dissection both is a cover, and consists of disjoint cubes

abbreviation *side* :: *axis* ⇒ *cube* ⇒ *real set* **where**
side ax c ≡ {*min ax c* ..< *max ax c*}

Sets of points generated from cubes

definition *to-set* :: *cube* ⇒ *point set* **where**
to-set c = {*p*. *px p* ∈ *side x c* ∧ *py p* ∈ *side y c* ∧ *pz p* ∈ *side z c*}
definition *bot* :: *cube* ⇒ *point set* **where**
bot c = {*p*. *px p* ∈ *side x c* ∧ *py p* ∈ *side y c* ∧ *pz p* = *min z c*}
definition *top* :: *cube* ⇒ *point set* **where**
top c = {*p*. *px p* ∈ *side x c* ∧ *py p* ∈ *side y c* ∧ *pz p* = *max z c*}

Moves a cube its width down (so top face to bottom face)

definition *shift-down* :: *cube* ⇒ *cube* **where**
shift-down c = *c* (| *point* := *point c* (| *pz* := *min z c* − *width c* |) |)

1.2 Calculations with sets from cubes

A bunch of statements we need about how cubes can be compared by *side*

lemma *top-shift-down-eq-bot*: *top* (*shift-down c*) = *bot c*
unfolding *top-def bot-def shift-down-def* **by** *simp*

Sets not empty

lemma *non-empty*: *is-valid c* ⟹ *to-set c* ≠ {}
unfolding *to-set-def* **by** *force*
lemma *top-non-empty*: *is-valid c* ⟹ *top c* ≠ {}
unfolding *top-def* **by** (*auto intro*!: *exI*[*of - point c* (| *pz* := *max z c* |)])

min of a cube is in corresponding *side*

lemma *min-in-side*: *is-valid c* ⟹ *min ax c* ∈ *side ax c*
by *simp*

lemma *min-ne-max*: *is-valid c* ⟹ *min ax c* ≠ *max ax c*
by *simp*
lemma *min-lt-max*: *is-valid c* ⟹ *min ax c* < *max ax c*
by *simp*

lemma *bot-subset*: *bot c* ⊆ *to-set c*
unfolding *bot-def to-set-def* **by**(*auto*)

1.2.1 Point membership

Points in a cube's set, by looking at membership of *side*

lemma *in-set-by-side*: *p* ∈ *to-set c* ⟷
px p ∈ *side x c* ∧ *py p* ∈ *side y c* ∧ *pz p* ∈ *side z c*
by(*cases p*, *simp add*: *to-set-def*)
lemma *in-set-by-side-2*: (|*px*=*x0*, *py*=*y0*, *pz*=*z0*|) ∈ *to-set c* ⟷
x0 ∈ *side x c* ∧ *y0* ∈ *side y c* ∧ *z0* ∈ *side z c*
by(*simp add*: *to-set-def*)

lemma *in-bot-by-side*: *p* ∈ *bot c* ⟷
px p ∈ *side x c* ∧ *py p* ∈ *side y c* ∧ *pz p* = *min z c*
by(*simp add*: *bot-def*)
lemma *in-bot-by-side-2*: (|*px*=*x0*, *py*=*y0*, *pz*=*z0*|) ∈ *bot c* ⟷
x0 ∈ *side x c* ∧ *y0* ∈ *side y c* ∧ *z0* = *min z c*
by(*simp add*: *in-bot-by-side*)

lemma *in-top-by-side*: *p* ∈ *top c* ⟷
px p ∈ *side x c* ∧ *py p* ∈ *side y c* ∧ *pz p* = *max z c*
by(*cases p*, *auto simp add*: *top-def*)
lemma *in-top-by-side-2*: (|*px*=*x0*, *py*=*y0*, *pz*=*z0*|) ∈ *top c* ⟷
x0 ∈ *side x c* ∧ *y0* ∈ *side y c* ∧ *z0* = *max z c*
by(*simp add*: *top-def*)

lemma *all-point-iff*: (∀ *p*. *P p*) ⟷ (∀ *x1 y1 z1*. *P* (|*px* = *x1*, *py* = *y1*, *pz* = *z1*|))

by (*metis point.cases*)

Intersection by *side*

lemma *set-intersect-by-side*: *to-set c1 ∩ to-set c2 ≠ {} ⟷*
side x c1 ∩ side x c2 ≠ {} ∧ side y c1 ∩ side y c2 ≠ {} ∧ side z c1 ∩ side z c2 ≠ {}
unfolding *set-eq-iff all-point-iff* **using** *in-set-by-side-2* **by** *blast*

lemma *bot-intersect-by-side*: *bot c1 ∩ bot c2 ≠ {}*
⟷ side x c1 ∩ side x c2 ≠ {} ∧ side y c1 ∩ side y c2 ≠ {} ∧ min z c1 = min z c2
proof(*intro iffI*)
assume *bot c1 ∩ bot c2 ≠ {}*
thus *side x c1 ∩ side x c2 ≠ {} ∧ side y c1 ∩ side y c2 ≠ {} ∧ min z c1 = min z c2*
using *in-bot-by-side* **by** *fastforce*
next
assume *side x c1 ∩ side x c2 ≠ {} ∧ side y c1 ∩ side y c2 ≠ {} ∧ min z c1 = min z c2*
thus *bot c1 ∩ bot c2 ≠ {}* **using** *in-bot-by-side-2* **by** *blast*
qed

lemma *bot-top-intersect-by-side*: *bot c1 ∩ top c2 ≠ {}*
⟷ side x c1 ∩ side x c2 ≠ {} ∧ side y c1 ∩ side y c2 ≠ {} ∧ min z c1 = max z c2
proof(*intro iffI*)
assume *bot c1 ∩ top c2 ≠ {}*
thus *side x c1 ∩ side x c2 ≠ {} ∧ side y c1 ∩ side y c2 ≠ {} ∧ min z c1 = max z c2*
using *in-bot-by-side in-top-by-side* **by** *fastforce*
next
assume *side x c1 ∩ side x c2 ≠ {} ∧ side y c1 ∩ side y c2 ≠ {} ∧ min z c1 = max z c2*
thus *bot c1 ∩ top c2 ≠ {}* **using** *in-bot-by-side-2 in-top-by-side-2* **by** *blast*
qed

1.2.2 Cubes subset of each other, by *side*

lemma *set-subset-by-side*: *to-set c1 ⊆ to-set c2 ⟷*
side x c1 ⊆ side x c2 ∧ side y c1 ⊆ side y c2 ∧ side z c1 ⊆ side z c2
unfolding *subset-eq Ball-def all-point-iff in-set-by-side* **by** *fastforce*
lemma *set-eq-by-side*: *to-set c1 = to-set c2 ⟷*
side x c1 = side x c2 ∧ side y c1 = side y c2 ∧ side z c1 = side z c2
using *set-subset-by-side* **by** *fast*

lemma *bot-eq-by-side*: *is-valid c1 ⟹ bot c1 = bot c2 ⟷*
side x c1 = side x c2 ∧ side y c1 = side y c2 ∧ min z c1 = min z c2
unfolding *bot-def set-eq-iff all-point-iff* **by** *fastforce*

lemma *bot-top-subset-by-side*: *is-valid c1* $\Longrightarrow$ *bot c1* $\subseteq$ *top c2* $\longleftrightarrow$
side x c1 $\subseteq$ *side x c2* $\wedge$ *side y c1* $\subseteq$ *side y c2* $\wedge$ *min z c1* = *max z c2*
unfolding *top-def bot-def subset-eq Ball-def all-point-iff* **by** *fastforce*
lemma *bot-top-eq-by-side*: *is-valid c1* $\Longrightarrow$ *bot c1* = *top c2* $\longleftrightarrow$
side x c1 = *side x c2* $\wedge$ *side y c1* = *side y c2* $\wedge$ *min z c1* = *max z c2*
unfolding *top-def bot-def set-eq-iff Ball-def all-point-iff* **by** *fastforce*

lemma *width-eq-if-side-eq*: $[\![$*is-valid c1*; *side ax c1* = *side ax c2*$]\!] \Longrightarrow$ *width c1* =
width c2
by(*simp add*:*Ico-eq-Ico*)

to-set is injective

lemma *to-set-inj*:
assumes *is-valid c1 to-set c1* = *to-set c2*
shows *c1* = *c2*
proof −
from *assms*(*2*) **have** ∗: *side x c1* = *side x c2 side y c1* = *side y c2 side z c1* =
side z c2
by(*metis set-eq-by-side*)+
hence *min x c1* = *min x c2 min y c1* = *min y c2 min z c1* = *min z c2*
by(*meson assms*(*1*) *Ico-eq-Ico less-add-same-cancel1*)+
hence *point c1* = *point c2* **by** *simp*
moreover from ∗ **have** *width c1* = *width c2* **by**(*metis assms*(*1*) *width-eq-if-side-eq*)
ultimately show *c1* = *c2* **by** *simp*
qed

Cube-Dissection.bot is also injective

lemma *bot-inj*: **assumes** *is-valid c1 bot c1* = *bot c2* **shows** *c1* = *c2*
proof −
from *assms* **have** *width c1* = *width c2* **by**(*metis assms bot-eq-by-side width-eq-if-side-eq*)
hence *to-set c1* = *to-set c2* **using** *assms bot-eq-by-side set-eq-by-side* **by** *auto*
thus *c1* = *c2* **by**(*metis assms*(*1*) *to-set-inj*)
qed

2 Cubing

We in this section introduce a dissection C of the unit cube

The cube we show there is no dissection of

definition *unit-cube* **where** *unit-cube* = (| *point*=(|*px*=*0*, *py*=*0*, *pz*=*0*|), *width*=*1* |)

lemma *min-unit-cube-0*: *min ax unit-cube* = *0*
by(*metis unit-cube-def axis.exhaust axis.simps*(*7*−*9*) *cube.select-convs*(*1*) *point.select-convs*(*1*−*3*))

lemma *unit-cube-valid*[*simp*]: *is-valid unit-cube*
by(*simp add*: *unit-cube-def*)

What we want to show doesn't exist. C is a set of cubes which satisfy:

1. All cubes are valid (width > 0)
2. All cubes a disjoint
3. The union of the cubes in C equal *unit-cube* (hence, all cubes are contained in *unit-cube*)
4. All cubes in C have different width
5. There are at least two cubes in C
6. There are a finite number of cubes in C

definition *is-dissection* :: *cube set* ⇒ *bool* **where**
is-dissection C ⟷
(∀ *c* ∈ *C*. *is-valid c*)
∧ *disjoint* (*image to-set C*)
∧ ⋃(*image to-set C*) = *to-set unit-cube*
∧ *inj-on width C* — All cubes are of different size
∧ *card C* ≥ *2* — At least two cubes
∧ *finite C*

From now on, C is some fixed dissection of *unit-cube*, and 'dissection' refers to this fact

context fixes *C* **assumes** *dissection*: *is-dissection C*
begin

2.1 Properties of *is-dissection*

lemma *valid-if-dissection*[*simp*]: *c* ∈ *C* ⟹ *is-valid c*
using *dissection* **unfolding** *is-dissection-def* **by** *fast*

lemma *side-unit-cube*:
side ax unit-cube = {*0*..<*1*}
by(*metis add.commute add.right-neutral cube.select-convs*(*2*) *min-unit-cube-0 unit-cube-def*)

lemma *subset-unit-cube-if-dissection*: *c* ∈ *C* ⟹ *to-set c* ⊆ *to-set unit-cube*
using *dissection* **unfolding** *is-dissection-def to-set-def* **by** *fast*

lemma *subset-unit-cube-by-side*:
c ∈ *C* ⟹ *side ax c* ⊆ {*0*..<*1*}
by(*metis*(*full-types*)*subset-unit-cube-if-dissection set-subset-by-side side-unit-cube axis.exhaust*)

lemma *eq-iff-intersect*: ⟦*c1* ∈ *C*; *c2* ∈ *C*⟧ ⟹ *c1* = *c2* ⟷ *to-set c1* ∩ *to-set c2* ≠ {}
using *dissection* **unfolding** *is-dissection-def disjoint-def*
by (*metis image-eqI inf.idem non-empty to-set-inj*)

Whenever we have a point in *unit-cube*, there exists a (unique) cube in C containing that point

lemma *obtain-cube*: $p \in$ *to-set unit-cube* $\Longrightarrow \exists\ c \in C.\ p \in$ *to-set* c
using *dissection* **unfolding** *is-dissection-def* **by** *blast*

If the top of c doesn't touch the top of *unit-cube*, then top of c must be covered by bottoms of cubes in C

lemma *top-cover-by-bot*:
assumes $c \in C$ *max z c* < 1
shows *top c* $\subseteq \bigcup$(*image bot C*)
proof(*intro subsetI*)
fix p **assume** *p-in-top*: $p \in$ *top c*
from *assms*(*1*) **have** *side x c* $\subseteq \{0..<1\}$ *side y c* $\subseteq \{0..<1\}$ *side z c* $\subseteq \{0..<1\}$
using *subset-unit-cube-by-side* **by** *blast*+
with *assms*(*2*) *p-in-top* **have** *px p* $\in \{0..<1\}$ *py p* $\in \{0..<1\}$ *pz p* $\in \{0..<1\}$
by(*simp add*: *in-top-by-side*)+
hence $p \in$ *to-set unit-cube* **using** *in-set-by-side side-unit-cube* **by** *blast*
then obtain c' **where** $c' \in C$ $p \in$ *to-set* c' **by**(*metis obtain-cube*)
moreover have $p \notin$ *to-set c* **using** *p-in-top in-top-by-side in-set-by-side* **by** *simp*
ultimately have $c' \neq c$ **by** *metis*
hence *to-set* $c' \cap$ *to-set* $c = \{\}$ **by**(*metis assms*(*1*) ‹$c' \in C$› *eq-iff-intersect*)
have $p \in$ *bot* c'
proof(*rule ccontr*)
assume $p \notin$ *bot* c'
hence *min z* $c' <$ *pz p pz p* $<$ *max z* c' **using** ‹$p \in$ *to-set* c'›
by(*simp-all add*: *in-set-by-side in-bot-by-side*)
hence *side z* $c' \cap$ *side z c* $\neq \{\}$ **using** ‹$p \in$ *top c*› **by**(*simp add*: *in-top-by-side*)
moreover have *side x c* $\cap$ *side x* $c' \neq \{\}$ *side y c* $\cap$ *side y* $c' \neq \{\}$
using ‹$p \in$ *top c*› ‹$p \in$ *to-set* c'› **by**(*simp-all add*: *in-top-by-side in-set-by-side assms*)
ultimately have *to-set* $c' \cap$ *to-set c* $\neq \{\}$ **using** *set-intersect-by-side* **by** *simp*
thus *False* **using** ‹*to-set* $c' \cap$ *to-set* $c = \{\}$› **by** *metis*
qed
thus $p \in \bigcup$(*image bot C*) **using** ‹$c' \in C$› **by** *blast*
qed

3 Hole

A hole h is a special kind of cube, where any cube whose bottom 'touches' the top of v must in fact have its bottom contained in the top of v. If $h \in C$, then this happens because all the other cubes surrounding h go up taller, forming a hole on top of v. Note that we don't require that $h \in C$, but this is only so we can prove that *unit-cube* shifted down by 1 is a hole - all other holes will in fact lie in C. The concept of a hole is inspired by the 'Valley' definition from [3]

3.1 Definitions

definition *is-hole* :: *cube* ⇒ *bool* **where**
is-hole h ⟷
is-valid h
∧ *top h* ⊆ ⋃(*image bot C*)
∧ (∀ *c* ∈ *C* . *bot c* ∩ *top h* ≠ {} ⟶ *bot c* ⊆ *top h*)
— *v* could be a cube in *C* (and most often is), but any other cube must be different width. Also, this assumption is not actually needed (as it follows from *v*, *c* ∈ *C*), but without it we have to do a special-case proof for the bottom of the *unit-cube*
∧ (∀ *c* ∈ *C* . *c* ≠ *h* ⟶ *width c* ≠ *width h*)

Subset of *C* which are on a given hole h

definition *is-on-hole* :: *cube* ⇒ *cube* ⇒ *bool* **where**
is-on-hole h c ⟷ *bot c* ⊆ *top h*
definition *filter-on-hole* :: *cube* ⇒ *cube set* **where**
filter-on-hole h = *Set.filter* (*is-on-hole h*) *C*

3.2 Properties of a hole

Terminology: 'on hole' means cube *c* with: *Cube-Dissection.bot c* ⊆ *Cube-Dissection.top h*. 'in hole' means point *p* with: *p* ∈ *Cube-Dissection.top h* h

local.filter-on-hole h ⊆ *C*

lemma *dissection-if-on-hole*[*simp*]: *c* ∈ *filter-on-hole h* ⟹ *c* ∈ *C*
unfolding *filter-on-hole-def* **by** *simp*

Holes, and cubes on them, are valid

lemma *valid-if-hole*[*simp*]: *is-hole h* ⟹ *is-valid h*
by(*metis is-hole-def*)
lemma *valid-if-on-hole*[*simp*]: *c* ∈ *filter-on-hole h* ⟹ *is-valid c*
by *simp*

lemma *on-hole-finite*: *is-hole h* ⟹ *finite* (*filter-on-hole h*)
by (*metis dissection is-dissection-def filter-on-hole-def finite-filter*)

lemma *on-hole-if-in-filter-on-hole*: *c* ∈ *filter-on-hole h* ⟹ *is-on-hole h c*
unfolding *filter-on-hole-def* **by** *simp*

lemma *on-hole-cover*: **assumes** *is-hole h* **shows** *top h* ⊆ ⋃(*image bot* (*filter-on-hole h*))
proof
fix *p* **assume** *p* ∈ *top h*
then obtain *c* **where** *c* ∈ *C p* ∈ *bot c* **using** *assms* **unfolding** *is-hole-def* **by** *blast*
with ‹*p* ∈ *top h*› **have** *bot c* ⊆ *top h* **using** *assms* **unfolding** *is-hole-def* **by** *blast*
hence *c* ∈ *filter-on-hole h*
by (*metis* ‹*c* ∈ *C*› *filter-on-hole-def is-on-hole-def member-filter*)

with ‹$p \in$ *bot* c› **show** $p \in \bigcup$(*image bot*(*filter-on-hole h*)) **by** *blast*
qed

Whenever we have a point p in the top of a hole h, there exists a (unique) cube $c \in$ *local.filter-on-hole h*, such that $p \in$ *Cube-Dissection.bot c*

lemma *obtain-cube-if-in-hole*: ⟦*is-hole h*; $p \in$ *top h*⟧
$\Longrightarrow \exists\, c \in$ *filter-on-hole h* . $p \in$ *bot c*
using *on-hole-cover* **by** *blast*

lemma *on-hole-inj-on-width*: *is-hole h* $\Longrightarrow$ *inj-on width* (*filter-on-hole h*)
using *dissection-if-on-hole inj-on-subset dissection*
is-dissection-def subset-iff **by** *metis*

3.3 Properties of cubes on a hole

lemma *neq-hole-if-on-hole*: $c \in$ *filter-on-hole h* $\Longrightarrow c \neq h$
using *valid-if-on-hole min-ne-max bot-top-subset-by-side is-on-hole-def*
on-hole-if-in-filter-on-hole **by** *blast*

lemma *subset-if-on-hole*: $c \in$ *filter-on-hole h* $\Longrightarrow$ *bot c* $\subseteq$ *top h*
unfolding *is-hole-def filter-on-hole-def is-on-hole-def* **by** *simp*

lemma *side-subset-if-on-hole*: ⟦$c \in$ *filter-on-hole h*; $ax \in \{x,y\}$⟧ $\Longrightarrow$ *side ax c* $\subseteq$ *side ax h*
using *bot-top-subset-by-side subset-if-on-hole* **by** *auto*

lemma *min-z-eq-max-z-hole-if-on-hole*:
$c \in$ *filter-on-hole h* $\Longrightarrow$ *min z c* = *max z h*
using *subset-if-on-hole* **by**(*simp add*: *bot-top-subset-by-side*)

lemma *z-eq-if-on-hole*:
⟦*c1* $\in$ *filter-on-hole h*; *c2* $\in$ *filter-on-hole h*⟧ $\Longrightarrow$ *min z c1* = *min z c2*
using *min-z-eq-max-z-hole-if-on-hole* **by** *simp*

Do not need to care about z-coordinate

lemma *eq-iff-side-eq-if-on-hole*: ⟦*c1* $\in$ *filter-on-hole h*; *c2* $\in$ *filter-on-hole h*⟧
$\Longrightarrow$ *c1* = *c2* $\longleftrightarrow$ *side x c1* = *side x c2* $\wedge$ *side y c1* = *side y c2*
by(*meson valid-if-on-hole z-eq-if-on-hole bot-eq-by-side bot-inj*)

Disjointness-lemmas:

lemma *eq-iff-bot-intersect-if-on-hole*:
assumes *c1* $\in$ *filter-on-hole h c2* $\in$ *filter-on-hole h*
shows *c1* = *c2* $\longleftrightarrow$ *bot c1* $\cap$ *bot c2* $\neq \{\}$
proof −
from *assms* **have** *c1* = *c2* $\longleftrightarrow$ *to-set c1* $\cap$ *to-set c2* $\neq \{\}$ **by**(*simp add*: *eq-iff-intersect*)
moreover have *min z c1* = *min z c2* **by**(*metis assms z-eq-if-on-hole*)
hence *to-set c1* $\cap$ *to-set c2* $\neq \{\} \longleftrightarrow$ *bot c1* $\cap$ *bot c2* $\neq \{\}$
using *set-intersect-by-side bot-intersect-by-side bot-subset subsetD* **by** *blast*

ultimately show *c1 = c2 ⟷ bot c1 ∩ bot c2 ≠ {}* **by** *blast*
qed

lemma *eq-iff-side-intersect-if-on-hole*:
⟦*c1 ∈ filter-on-hole h*; *c2 ∈ filter-on-hole h*⟧
⟹ c1 = c2 ⟷ side x c1 ∩ side x c2 ≠ {} ∧ side y c1 ∩ side y c2 ≠ {}
by(*meson z-eq-if-on-hole eq-iff-bot-intersect-if-on-hole bot-intersect-by-side*)

lemma *width-on-hole-lt-width-hole*:
assumes *is-hole h c ∈ filter-on-hole h* **shows** *width c < width h*
proof(*rule ccontr*)
assume *¬width c < width h*
hence *width c ≥ width h* **by** *simp*
moreover have *c ≠ h* **by** (*metis assms*(*2*) *neq-hole-if-on-hole*)
hence *width c ≠ width h* **using** *assms* **unfolding** *is-hole-def* **by** *simp*
ultimately have *width c > width h* **by** *simp*
hence *¬*(*side x c ⊆ side x h*) **using** *assms less-le-not-le* **by** *fastforce*
hence *¬*(*bot c ⊆ top h*) **using** *assms* **by**(*auto simp add*: *bot-top-subset-by-side*)
moreover have *bot c ⊆ top h* **by**(*metis assms*(*2*) *subset-if-on-hole*)
ultimately show *False* **by** *metis*
qed

lemma *strict-subset-if-on-hole*: **assumes** *is-hole h c ∈ filter-on-hole h*
shows *bot c ⊂ top h*
proof
show *bot c ⊆ top h* **by**(*metis assms*(*2*) *subset-if-on-hole*)
next
have *width c ≠ width h* **using** *assms width-on-hole-lt-width-hole* **by** *fastforce*
hence *side x c ≠ side x h* **by**(*metis assms*(*1*) *valid-if-hole width-eq-if-side-eq*)
thus *bot c ≠ top h* **using** *assms bot-top-eq-by-side* **by** *simp*
qed

lemma *on-hole-non-empty*: *is-hole h ⟹ filter-on-hole h ≠ {}*
using *on-hole-cover*
by (*metis Sup-empty image-empty subset-empty top-non-empty valid-if-hole*)

4 Bottom of *unit-cube* is a hole

lemma *bot-unit-cube-cover-by-bot*: *bot unit-cube ⊆ ⋃*(*image bot C*)
proof
fix *p*
assume *p ∈ bot unit-cube*
hence *p ∈ to-set unit-cube* **using** *bot-subset* **by** *auto*
then obtain *c* **where** *c ∈ C p ∈ to-set c* **by**(*metis obtain-cube*)
from ‹*p ∈ bot unit-cube*› **have** *pz p = 0* **by**(*metis min-unit-cube-0 in-bot-by-side*)
hence *min z c ≤ 0* **using** ‹*p ∈ to-set c*› **by**(*simp add*: *in-set-by-side*)
moreover have *side z c ⊆ {0..<1}* **by**(*metis ‹c ∈ C› subset-unit-cube-by-side*)
hence *min z c ≥ 0* **using** ‹*c ∈ C*› **by** *fastforce*
ultimately have *min z c = 0* **by** *simp*

hence $p \in bot\ c$ **using** ‹$p \in to\text{-}set\ c$› ‹$pz\ p = 0$› **by**(*simp add*: *in-set-by-side in-bot-by-side*)
thus $p \in \bigcup\ (image\ bot\ C)$ **using** ‹$c \in C$› **by** *auto*
qed

lemma *eq-if-width-eq-if-subset*:
assumes $width\ c1 = width\ c2\ to\text{-}set\ c1 \subseteq to\text{-}set\ c2$
shows $to\text{-}set\ c1 = to\text{-}set\ c2$
proof−
from *assms*(*2*) **have** $side\ x\ c1 \subseteq side\ x\ c2\ side\ y\ c1 \subseteq side\ y\ c2\ \ side\ z\ c1 \subseteq side\ z\ c2$
using *set-subset-by-side* **by** *blast*+
hence $side\ x\ c1 = side\ x\ c2\ side\ y\ c1 = side\ y\ c2\ side\ z\ c1 = side\ z\ c2$
using *assms*(*1*) **by** *fastforce*+
thus $to\text{-}set\ c1 = to\text{-}set\ c2$ **using** *set-eq-by-side* **by** *blast*
qed

lemma *width-ne-one*:
assumes $c \in C$
shows $width\ c \neq 1$
proof(*rule ccontr*)
obtain c' **where** $c' \in C\ c \neq c'$ **using** *assms dissection*
unfolding *is-dissection-def* **by**(*auto simp add*: *card-le-Suc-iff numeral-eq-Suc*)
hence $to\text{-}set\ c' \cap to\text{-}set\ c = \{\}$ **using** *assms* **by**(*metis eq-iff-intersect*)
assume ¬ $width\ c \neq 1$
hence $width\ c = 1$ **by** *blast*
moreover have $to\text{-}set\ c \subseteq to\text{-}set\ unit\text{-}cube$ **by**(*metis assms subset-unit-cube-if-dissection*)
ultimately have $to\text{-}set\ c = to\text{-}set\ unit\text{-}cube$ **using** *eq-if-width-eq-if-subset*
unfolding *unit-cube-def* **by** *auto*
with ‹$c' \in C$› **have** $to\text{-}set\ c' \subseteq to\text{-}set\ c$ **by**(*metis subset-unit-cube-if-dissection*)
hence $to\text{-}set\ c' \cap to\text{-}set\ c \neq \{\}$ **using** ‹$c' \in C$› **by**(*simp add*: *non-empty Int-absorb2*)
moreover have $to\text{-}set\ c' \cap to\text{-}set\ c = \{\}$ **by**(*metis assms* ‹$c' \in C$› ‹$c \neq c'$› *eq-iff-intersect*)
ultimately show *False* **by** *metis*
qed

Combines the previous lemmas, to show that the bottom of *unit-cube* is a hole

proposition *hole-unit-cube*: *is-hole* (*shift-down unit-cube*)
proof−
let *?b* = *shift-down unit-cube* — cube with *point* (0,0,-1) and *width* 1
show *?thesis* **unfolding** *is-hole-def*
proof(*intro conjI subsetI ballI impI*)
show *is-valid ?b* **by**(*simp add*: *shift-down-def*)
next
fix *p* **assume** $p \in top\ ?b$
hence $p \in bot\ unit\text{-}cube$ **by**(*simp add*: *top-shift-down-eq-bot*)
thus $p \in \bigcup(image\ bot\ C)$ **using** *bot-unit-cube-cover-by-bot* **by** *blast*

next
fix *c p* **assume** *c* ∈ *C bot c* ∩ *top ?b* ≠ {} *p* ∈ *bot c*
with ‹*bot c* ∩ *top ?b* ≠ {}› **have** *min z c* = *0*
by(*metis top-shift-down-eq-bot bot-intersect-by-side min-unit-cube-0*)
hence *pz p* = *0* **by**(*metis* ‹*p* ∈ *bot c*› *in-bot-by-side*)
moreover from ‹*p* ∈ *bot c*› **have** *px p* ∈ {*0*..<*1*} *py p* ∈ {*0*..<*1*}
using ‹*c* ∈ *C*› *subset-unit-cube-by-side in-bot-by-side* **by** *blast+*
ultimately have *p* ∈ *bot unit-cube* **by**(*metis min-unit-cube-0 in-bot-by-side side-unit-cube*)
thus *p* ∈ *top ?b* **by**(*metis top-shift-down-eq-bot*)
next
fix *c* **assume** *c* ∈ *C c* ≠ *?b*
hence *width c* ≠ *1* **by**(*metis width-ne-one*)
thus *width c* ≠ *width ?b* **by**(*simp add*: *unit-cube-def shift-down-def*)
qed
qed

5 Minimum cube on hole is interior

context
fixes *h* **assumes** *hole*: *is-hole h*
begin

For this section, we fix a hole *h*, and define *cmin* to be the smallest cube on this hole. Theorem *hole* refers to this fact. The goal of this section is then to show that *cmin* itself is a hole.

5.1 Definition: Minimum cube on *h*

cmin is the smallest cube on the hole *h*

definition *cmin*:: *cube*
where *cmin* = (*ARG-MIN width c* . *c* ∈ *filter-on-hole h*)

lemma *arg-min-exist*: ⟦*finite C'*; *C'* ≠ {}⟧ ⟹ (*ARG-MIN width c* . *c* ∈ *C'*) ∈ *C'*
by(*metis arg-min-on-def arg-min-if-finite*(*1*))

This lemma also shows that *local.cmin* exists

lemma *cmin-on-h*: *cmin* ∈ *filter-on-hole h*
unfolding *cmin-def* **by**(*metis hole arg-min-exist on-hole-finite on-hole-non-empty*)

lemma *cmin-valid*[*simp*]: *is-valid cmin*
by(*metis cmin-on-h valid-if-on-hole*)

lemma *arg-min-minimal*: ⟦*finite C'*; *c* ∈ *C'*⟧ ⟹ *width* (*ARG-MIN width c* . *c* ∈ *C'*) ≤ *width c*
by(*metis arg-min-least arg-min-on-def empty-iff*)

lemma *cmin-minimal*: $c \in$ *filter-on-hole* $h \Longrightarrow$ *width cmin* $\leq$ *width* c
unfolding *cmin-def* **by**(*metis hole arg-min-minimal on-hole-finite*)

lemma *cmin-minimal-strict*:
assumes $c \in$ *filter-on-hole* h $c \neq$ *cmin*
shows *width cmin* $<$ *width* c
proof −
have *width cmin* $\leq$ *width* c **using** *assms*(*1*) **by**(*rule cmin-minimal*)
moreover have *width cmin* $\neq$ *width* c
by (*metis assms on-hole-inj-on-width assms*(*2*) *cmin-on-h hole inj-on-contraD*)
ultimately show *width cmin* $<$ *width* c **by** *simp*
qed

lemma *cmin-max-z-neq-one*: *max z cmin* $<$ *1*
proof −
let *?On-h* = *filter-on-hole h*
have *bot cmin* $\subset$ *top h* **by**(*metis hole cmin-on-h strict-subset-if-on-hole*)
then obtain c **where** $c \in$ *?On-h* $c \neq$ *cmin* **using** *hole obtain-cube-if-in-hole* **by** *blast*
hence *min z cmin* = *min z c* **by**(*metis cmin-on-h* ‹$c \in$ *?On-h*› *z-eq-if-on-hole*)
hence *max z cmin* $<$ *max z c* **by**(*simp add*: ‹$c \in$ *?On-h*› ‹$c \neq$ *cmin*› *cmin-minimal-strict*)
moreover have *side z c* $\subseteq \{0..<1\}$
by(*metis* ‹$c \in$ *?On-h*› *dissection-if-on-hole subset-unit-cube-by-side*)
hence *max z c* $\leq$ *1* **using** ‹$c \in$ *?On-h*› *valid-if-on-hole* **by** (*simp add*: *less-le-not-le*)
ultimately show *max z cmin* $<$ *1* **by** *simp*
qed

5.2 Minimum cube on hole is interior

All squares on the boundary of h

definition *is-on-boundary* :: *axis* ⇒ *cube* ⇒ *bool* **where**
is-on-boundary ax c ⟷ *min ax h* = *min ax c* ∨ *max ax h* = *max ax c*

Shows that IF *local.cmin* is on a boundary *ax*, then we find some *ax*-coordinate r, which is further from the boundary than the edge of *local.cmin*, but closer than the edge of any other cube sufficiently close to the boundary.

lemma *cmin-on-boundary*:
assumes *is-on-boundary ax cmin* $ax \in \{x, y\}$
shows $\exists\, r$.
$r \in$ (*side ax h* − (*side ax cmin*)) ∧
(∀ $c \in$ *filter-on-hole h* . $c \neq$ *cmin* ⟶ *side ax cmin* ∩ *side ax c* ≠ {} ⟶ $r \in$ *side ax c*)
proof −
let *?On-h* = *filter-on-hole h*
let *?c-2nd-min* = *ARG-MIN width c* . $c \in$ (*Set.remove cmin ?On-h*)
have *bot cmin* $\subset$ *top h* **by**(*metis hole cmin-on-h strict-subset-if-on-hole*)
hence *Set.remove cmin ?On-h* ≠ {} **using** *hole obtain-cube-if-in-hole* **by** *fastforce*
moreover have *finite* (*Set.remove cmin ?On-h*)
by (*metis hole on-hole-finite remove-def finite-Diff*)

ultimately have *?c-2nd-min ∈ Set.remove cmin ?On-h* **by**(*metis arg-min-exist*)
hence *?c-2nd-min ∈ ?On-h* **and** *width cmin < width ?c-2nd-min*
by(*simp*, *simp add*: *cmin-minimal-strict*)
then obtain *ε* **where** *ε > 0 width cmin + ε < width ?c-2nd-min*
by(*metis field-le-epsilon less-le-not-le nle-le*)
have *ε-prop*: *∀ c ∈ ?On-h . cmin ≠ c ⟶ width cmin + ε < width c*
proof(*intro ballI impI*)
fix *c* **assume** *c ∈ ?On-h cmin ≠ c*
hence *c ∈ Set.remove cmin ?On-h* **by** *auto*
hence *width ?c-2nd-min ≤ width c*
using ‹*Set.remove cmin ?On-h ≠ {}*› ‹*finite* (*Set.remove cmin ?On-h*) › *arg-min-minimal*
by *blast*
thus *width cmin + ε < width c* **using** ‹*width cmin + ε < width ?c-2nd-min*› **by** *simp*
qed
then show *?thesis* **using** *assms*(*1*) **unfolding** *is-on-boundary-def*
proof(*elim disjE*)
assume *min ax h = min ax cmin*
let *?r = max ax cmin + ε*
show *?thesis*
proof(*intro exI conjI ballI impI*)
have *?r < min ax cmin + width ?c-2nd-min* **using** ‹*width cmin + ε < width ?c-2nd-min*› **by** *auto*
also have *... = min ax h + width ?c-2nd-min* **using** ‹*min ax h = min ax cmin*› **by** *simp*
finally have *?r < max ax h*
using ‹*?c-2nd-min ∈ ?On-h*› *hole width-on-hole-lt-width-hole* **by** *fastforce*
moreover have *min ax cmin < max ax cmin* **using** *assms*(*1*) **by**(*simp add*: *min-lt-max*)
hence *min ax h < ?r* **using** ‹*ε > 0*› ‹*min ax h = min ax cmin*› **by** *linarith*
ultimately have *?r ∈ side ax h* **by** *simp*
moreover from ‹*ε > 0*› **have** *?r ∉ side ax cmin* **by** *simp*
ultimately show *?r ∈ side ax h − side ax cmin* **by** *simp*
next
fix *c* **assume** *c ∈ ?On-h c ≠ cmin side ax cmin ∩ side ax c ≠ {}*
from ‹*side ax cmin ∩ side ax c ≠ {}*› **have** *min ax c < max ax cmin* **by** *auto*
also have *... < ?r* **using** ‹*ε>0*› **by** *simp*
finally have *min ax c ≤ ?r* **by** *simp*
moreover have *bot c ⊆ top h* **by**(*metis* ‹*c ∈ ?On-h*› *subset-if-on-hole*)
hence *side ax c ⊆ side ax h* **using** ‹*c ∈ ?On-h*› *assms*(*2*) *bot-top-subset-by-side* **by** *fastforce*
hence *min ax h ≤ min ax c* **using** ‹*c ∈ ?On-h*› **by** *fastforce*
hence *min ax cmin ≤ min ax c* **using** ‹*min ax h = min ax cmin*› **by** *simp*
hence *?r < max ax c* **using** ‹*c ∈ ?On-h*› ‹*c ≠ cmin*› *ε-prop* **by** *auto*
ultimately show *?r ∈ side ax c* **by** *simp*
qed
next
assume *max ax h = max ax cmin*

```
    let ?r = min ax cmin − ε
    show ?thesis
    proof(intro exI conjI ballI impI)
      have min ax h = max ax h − width h by simp
      also have ... < max ax h − width ?c-2nd-min
        using hole width-on-hole-lt-width-hole ‹?c-2nd-min ∈ ?On-h› by simp
      also have ... = max ax cmin − width ?c-2nd-min using ‹max ax h = max
ax cmin› by simp
      also have ... < ?r using ‹width cmin + ε < width ?c-2nd-min› by argo
      finally have min ax h < ?r by simp
      moreover have min ax cmin < max ax cmin using assms(1) by(simp add:
min-lt-max)
      hence ?r < max ax h using ‹max ax h = max ax cmin› ‹ε > 0› by argo
      ultimately have ?r ∈ side ax h by simp
      moreover have ?r ∉ side ax cmin using ‹ε > 0› by simp
      ultimately show ?r ∈ side ax h − side ax cmin by simp
    next
      fix c assume c ∈ ?On-h c ≠ cmin side ax cmin ∩ side ax c ≠ {}
      have ?r < min ax cmin using ‹ε>0› by simp
      also have ... < max ax c using ‹side ax cmin ∩ side ax c ≠ {}› by simp
      finally have ?r < max ax c by simp
      moreover have bot c ⊆ top h by(metis ‹c ∈ ?On-h› subset-if-on-hole)
      hence side ax c ⊆ side ax h
        using ‹c ∈ ?On-h› assms(2) bot-top-subset-by-side by fastforce
      hence max ax c ≤ max ax h using ‹c ∈ ?On-h› valid-if-on-hole by fastforce
      hence max ax c ≤ max ax cmin using ‹ max ax h = max ax cmin› by simp
      hence min ax c≤ ?r using ‹c ∈ ?On-h› ‹c ≠ cmin› ε-prop by auto
      ultimately show ?r ∈ side ax c by simp
    qed
  qed
qed
```

Using the previous lemma, we show that *local.cmin* being on the boundary leads to a contradiction

```
lemma cmin-not-on-boundary-by-axis:
  assumes ax ∈ {x, y}
  shows ¬is-on-boundary ax cmin
proof(rule ccontr, unfold not-not)
  let ?On-h = filter-on-hole h
  assume is-on-boundary ax cmin
  then obtain r where r ∈ (side ax h − side ax cmin) and
    r-prop: (∀ c ∈ ?On-h . c ≠ cmin ⟶ side ax cmin ∩ side ax c ≠ {} ⟶ r ∈
side ax c)
    by(metis assms cmin-on-boundary)
  hence r ∈ side ax h and r ∉ side ax cmin by blast+
  let ?ax2 = if ax = x then y else x
  let ?p1 = if ax = x then (|px=r, py=min y cmin, pz=max z h|) else (|px=min x
cmin, py=r, pz=max z h|)
  have side ?ax2 cmin ⊆ side ?ax2 h using cmin-on-h side-subset-if-on-hole by
```

simp
hence *min ?ax2 cmin ∈ side ?ax2 h* **using** *min-in-side* **by** *fastforce*
hence *?p1 ∈ top h* **using** ‹*r ∈ side ax h*› *in-top-by-side assms* **by** (*simp split*: *if-splits*)
then obtain *c1* **where** *?p1 ∈ bot c1* **and** *c1-on-h*: *c1 ∈ ?On-h*
using *hole obtain-cube-if-in-hole* **by** *blast*
hence *cmin ≠ c1* **using** ‹*r ∉ side ax cmin*› *assms in-bot-by-side* **by** (*auto split*: *if-splits*)
hence *width cmin < width c1* **by**(*metis* ‹*c1 ∈ ?On-h*› *cmin-minimal-strict*)
hence ¬ *side ?ax2 c1 ⊆ side ?ax2 cmin* **using** *c1-on-h valid-if-on-hole* **by** *fastforce*
then obtain *s* **where** *s ∈ side ?ax2 c1 s ∉ side ?ax2 cmin* **by** *blast*
let *?p2=if ax = x then* (|*px=min x cmin*, *py=s*, *pz=max z h*|) *else* (|*px=s*, *py=min y cmin*, *pz=max z h*|)
have *side ax cmin ⊆ side ax h* **using** *assms cmin-on-h side-subset-if-on-hole* **by** *blast*
hence *min ax cmin ∈ side ax h* **using** *min-in-side* **by** *fastforce*
moreover have *side ?ax2 c1 ⊆ side ?ax2 h* **using** ‹*c1 ∈?On-h*› *side-subset-if-on-hole* **by** *simp*
hence *s ∈ side ?ax2 h* **using** ‹*s ∈ side ?ax2 c1*› **by** *blast*
ultimately have *?p2 ∈ top h* **using** *in-top-by-side assms* **by** (*simp split*: *if-splits*)
then obtain *c2* **where** *?p2 ∈ bot c2* **and** *c2-on-h*: *c2 ∈ ?On-h*
using *hole obtain-cube-if-in-hole* **by** *blast*
let *?p3 = if ax = x then* (|*px=r*, *py=s*, *pz=max z h*|) *else* (|*px=s*, *py=r*, *pz=max z h*|)
— We show that *if ax = x then* (|*px = r*, *py = s*, *pz = max z h*|) *else* (|*px = s*, *py = r*, *pz = max z h*|) is in the intersection of *c1* and *c2*, but *p2* is only in *c2*, contradicting disjointness
have *?p2 ∉ bot cmin* **using** ‹*s ∉ side ?ax2 cmin*› *in-bot-by-side* **by** (*auto split*: *if-splits*)
hence *cmin ≠ c2* **using** ‹*?p2 ∈ bot c2*› **by** *blast*
hence *r ∈ side ax c2* **using** *r-prop c2-on-h min-in-side* ‹*?p2 ∈ bot c2*› *in-bot-by-side assms*
by *fastforce*
hence *?p3 ∈ bot c2* **using** ‹*?p2 ∈ bot c2*› *in-bot-by-side assms* **by** (*simp split*: *if-splits*)
moreover have *?p3 ∈ bot c1* **using** ‹*?p1 ∈ bot c1*› ‹*s ∈ side ?ax2 c1*› *in-bot-by-side* **by** *auto*
ultimately have *c1 = c2* **using** *c1-on-h c2-on-h eq-iff-bot-intersect-if-on-hole*
by *blast*
moreover from ‹*s ∉ side ?ax2 cmin*› **have** *?p2 ∉ to-set cmin* **using** *in-set-by-side* **by** *auto*
hence *cmin ≠ c2* **using** *eq-iff-side-eq-if-on-hole c2-on-h* ‹*?p2 ∈ bot c2*› *bot-subset*

by *blast*
hence *to-set cmin ∩ to-set c2 = {}* **using** *cmin-on-h c2-on-h eq-iff-intersect* **by** *simp*
hence *side ?ax2 cmin ∩ side ?ax2 c2 = {}* **using** ‹*?p2 ∈ bot c2*› ‹*cmin ≠ c2*›
c2-on-h assms cmin-on-h in-bot-by-side eq-iff-side-intersect-if-on-hole **by** *fast-*

force
hence *min ?ax2 cmin ∉ side ?ax2 c2* **using** *cmin-valid min-in-side* **by** *blast*
hence *?p1 ∉ bot c2* **using** *assms in-bot-by-side* **by** *fastforce*
hence *c1 ≠ c2* **using** ‹*?p1 ∈ bot c1*› **by** *blast*
ultimately show *False* **by** *blast*
qed

Previous result, written as inequalities instead

proposition *cmin-not-on-boundary*:
min x h < min x cmin ∧ max x cmin < max x h
∧ min y h < min y cmin ∧ max y cmin < max y h
proof−
have *side x cmin ⊆ side x h ∧ side y cmin ⊆ side y h*
using *cmin-on-h side-subset-if-on-hole* **by** *blast*
hence *min x h ≤ min x cmin ∧ max x cmin ≤ max x h ∧*
min y h ≤ min y cmin ∧ max y cmin ≤ max y h **using** *cmin-valid* **by** *auto*
thus *min x h < min x cmin ∧ max x cmin < max x h ∧*
min y h < min y cmin ∧ max y cmin < max y h
using *cmin-not-on-boundary-by-axis* **unfolding** *is-on-boundary-def* **by** *force*
qed

6 Minimum cube of hole induces hole on top

The main result of this proof - the minimum cube on a hole is itself a hole!

proposition *hole-cmin*:
shows *is-hole cmin*
proof−
let *?On-cmin = filter-on-hole cmin*
show *?thesis* **unfolding** *is-hole-def*
proof(*intro conjI ballI impI*)
show *is-valid cmin* **by** *simp*
next
show *top cmin ⊆ ⋃(image bot C)*
by(*metis top-cover-by-bot cmin-max-z-neq-one cmin-on-h dissection-if-on-hole*)
next
fix *c* **assume** *c ∈ C c ≠ cmin*
thus *width c ≠ width cmin*
by(*metis dissection dissection-if-on-hole cmin-on-h inj-onD is-dissection-def*)
next
fix *c* **assume** *c ∈ C bot c ∩ top cmin ≠ {}*
hence *c-valid*: *is-valid c* **by** *simp*
show *bot c ⊆ top cmin*
proof(*rule ccontr*)
assume *¬ bot c ⊆ top cmin*
from ‹*bot c ∩ top cmin ≠ {}*› **have** *min z c = max z cmin* **using** *bot-top-intersect-by-side*

by *blast*
with ‹*¬ bot c ⊆ top cmin*› **obtain** *ax* **where** *ax*: *ax ∈*{*x*,*y*} *¬side ax c ⊆*

side ax cmin
using *bot-top-subset-by-side c-valid* **by** (*meson insert-iff*)
let *?ax2 = if ax=x then y else x*
have *?ax2* ∈ {*x*,*y*} **by** *simp*
have *min ax h < min ax cmin max ax cmin < max ax h*
using *ax(1) cmin-not-on-boundary* **by** *blast+*
moreover have *side ax c* ∩ *side ax cmin* ≠ {}
using ‹*bot c* ∩ *top cmin* ≠ {}› *ax(1) bot-top-intersect-by-side* **by** *blast*
ultimately have *side ax c* ∩ *side ax h* − *side ax cmin* ≠ {} **using** *ax(2)* **by** *fastforce*
then obtain *a* **where** *a-prop*: *a* ∈ *side ax c* ∩ *side ax h a* ∉ *side ax cmin* **by** *blast*
moreover from ‹*bot c* ∩ *top cmin* ≠ {}› **have** *side ?ax2 c* ∩ *side ?ax2 cmin* ≠ {}
using *bot-top-intersect-by-side* ‹*?ax2* ∈ {*x*,*y*}› **by** *simp*
then obtain *b* **where** *b-prop*: *b* ∈ *side ?ax2 c* ∩ *side ?ax2 cmin* **by** *blast*
hence *b-prop-2*: *b* ∈ *side ?ax2 c* ∩ *side ?ax2 h*
using ‹*?ax2* ∈ {*x*,*y*}› *cmin-on-h side-subset-if-on-hole* **by** *blast*
let *?x0 = if ax=x then a else b* **and** *?y0 = if ax=y then a else b*
have *xy0-props*: *?x0* ∈ *side x c* ∩ *side x h ?y0* ∈ *side y c* ∩ *side y h*
?x0 ∉ *side x cmin* ∨ *?y0* ∉ *side y cmin*
proof −
show*?x0* ∈ *side x c* ∩ *side x h* **using** *a-prop b-prop ax(1)* ‹*?ax2* ∈ {*x*,*y*}›
using *b-prop-2* **by** *presburger*
show *?y0* ∈ *side y c* ∩ *side y h* **using** *a-prop b-prop-2 ax(1)* **by** *fastforce*
show *?x0* ∉ *side x cmin* ∨ *?y0* ∉ *side y cmin* **using** *a-prop ax(1)* **by** *fastforce*
qed
let *?pd* = (|*px=?x0*, *py=?y0*, *pz=max z h*|)
let *?pu* = (|*px=?x0*, *py=?y0*, *pz=min z c*|)
have *max z h = min z cmin* **by**(*metis cmin-on-h min-z-eq-max-z-hole-if-on-hole*)
hence *?pd* = (|*px=?x0*, *py=?y0*, *pz=min z cmin*|) **by** *simp*
moreover have *min z cmin < min z c* **using** *min-lt-max* ‹*min z c = max z cmin*› **by** *simp*
hence *min z cmin* ∉ *side z c* **using** *in-set-by-side* **by** *simp*
ultimately have *?pd* ∉ *to-set c* **by**(*simp add*: *in-set-by-side*)
have *?pu* ∈ *bot c* **using** *c-valid xy0-props in-bot-by-side* **by** *simp*
hence *?pu* ∈ *to-set c* **using** *bot-subset* **by** *blast*
have *?pd* ∉ *bot cmin* **using** ‹*?x0* ∉ *side x cmin* ∨ *?y0* ∉ *side y cmin*› *in-bot-by-side* **by** *auto*
have *?pd* ∈ *top h* **using** *valid-if-hole xy0-props in-top-by-side* **by** *simp*
then obtain *c′* **where** *c′* ∈ *filter-on-hole h ?pd* ∈ *bot c′*
using *hole obtain-cube-if-in-hole* **by** *blast*
hence *c′* ≠ *cmin* **using** ‹*?pd* ∉ *bot cmin*› **by** *blast*
hence *width cmin < width c′*
using ‹*c′* ∈ *filter-on-hole h*› *cmin-minimal-strict* **by** *simp*
moreover have *min z cmin = min z c′*
by(*metis cmin-on-h* ‹*c′* ∈ *filter-on-hole h*› *z-eq-if-on-hole*)
ultimately have *max z cmin < max z c′* **by** *simp*

hence *min z c < max z c'* **by**(*metis* ‹*min z c = max z cmin*›)
hence *min z c ∈ side z c'* **using** ‹*min z cmin < min z c*› ‹*min z cmin = min z c'*› **by** *auto*
hence *?pu ∈ to-set c'* **using** ‹*c' ∈ filter-on-hole h*› ‹*?pd ∈ bot c'*›
in-bot-by-side in-set-by-side ‹*min z c = max z cmin*› **by** *simp*
hence *to-set c ∩ to-set c' ≠ {}* **using** ‹*?pu ∈ to-set c*› **by** *blast*
hence *c = c'* **using** ‹*c' ∈ filter-on-hole h*› ‹*c ∈ C*› *eq-iff-intersect* **by** *auto*
moreover have *c ≠ c'* **using** ‹*?pd ∉ to-set c*› ‹*?pd ∈ bot c'*› **using** *bot-subset* **by** *blast*
ultimately show *False* **by** *blast*
qed
qed
qed

The main purpose of the previous result: From the proposition, *hole-cmin* when given the hole h induce another hole h' (i.e., *local.cmin*), which is in C and is strictly smaller.

lemma *recursive-step*: $\exists\, h'.\ h' \in C \wedge$ *is-hole* $h' \wedge$ *width* $h' <$ *width* h
proof−
show *?thesis*
proof(*intro exI conjI*)
show *cmin ∈ C* **using** *cmin-on-h* **by** *force*
show *is-hole cmin* **by**(*simp add*: *hole-cmin*)
show *width cmin < width h* **by**(*simp add*: *hole cmin-on-h width-on-hole-lt-width-hole*)
qed
qed

Here we end the context in which h is some fixed hole (and hence also the specific *local.cmin*)

end

7 The main result

We combine the previous lemmas inductively as follows: 0: Start with the bottom of *unit-cube*, which we showed is a hole. n: For each hole, take the minimum cube on this hole, which is then a new hole, strictly smaller, and in C. Hence, C is infinite.

definition *next-hole*:: *cube* ⇒ *cube* **where**
next-hole h = (*SOME h'* . *h' ∈ C ∧ is-hole h' ∧ width h' < width h*)

lemma *next-hole-exist*: *is-hole h*
⟹ *next-hole h ∈ C ∧ is-hole* (*next-hole h*) ∧ *width* (*next-hole h*) < *width h*
unfolding *next-hole-def* **by** (*rule someI-ex*) (*rule recursive-step*)

For following proof, we want the image of *nth-hole* to be contained in C, hence we start at *1* (= *Suc 0*). *nth-hole* is a function from $\mathbb{N}$ to C

definition *nth-hole* :: *nat* ⇒ *cube* **where**

nth-hole n = (*next-hole* ^^ *Suc n*) (*shift-down unit-cube*)

Each cube in the image of *local.nth-hole* is a hole

lemma *nth-hole-is-hole*: *is-hole* (*nth-hole n*)
proof(*induction n*)
case *0*
have *nth-hole 0* = *next-hole* (*shift-down unit-cube*) **by**(*simp add*: *nth-hole-def*)
moreover have *is-hole* (*shift-down unit-cube*) **by**(*rule hole-unit-cube*)
ultimately show *?case* **by**(*simp add*: *next-hole-exist*)
next
case (*Suc n*)
then show *?case* **by**(*simp add*: *nth-hole-def next-hole-exist*)
qed

uminus is *uminus*, and *strict-mono* means strictly increasing (not strictly monotonous, as the name might suggest)

lemma *nth-hole-strict-decreasing*: *strict-mono* (*uminus* ∘ *width* ∘ *nth-hole*)
proof(*rule iffD2*, *rule strict-mono-Suc-iff*, *intro allI*)
fix *n*
have *is-hole* (*nth-hole n*) **by** (*rule nth-hole-is-hole*)
thus (*uminus* ∘ *width* ∘ *nth-hole*) *n* < (*uminus* ∘ *width* ∘ *nth-hole*) (*Suc n*)
by(*simp add*: *nth-hole-def next-hole-exist*)
qed

local.nth-hole is injective

lemma *nth-hole-inj* : *inj nth-hole*
proof −
have *strict-mono* (*uminus* ∘ *width* ∘ *nth-hole*) **by**(*rule nth-hole-strict-decreasing*)
hence *inj* (*uminus* ∘ *width* ∘ *nth-hole*) **by**(*simp add*: *strict-mono-imp-inj-on*)
thus *inj nth-hole* **by**(*simp add*:*inj-on-imageI2*)
qed

The image (range) of *local.nth-hole* is contained in *C*

lemma *nth-hole-in*: *nth-hole n* ∈ *C*
proof(*cases n*)
case *0*
then show *?thesis* **by**(*simp add*: *hole-unit-cube nth-hole-def next-hole-exist*)
next
case (*Suc nat*)
then obtain *m* **where** *n* = *Suc m* **by** *simp*
have *is-hole* (*nth-hole m*) **by**(*simp add*: *nth-hole-is-hole*)
hence *next-hole* (*nth-hole m*) ∈ *C* **by**(*simp add*: *next-hole-exist*)
thus *nth-hole n* ∈ *C* **by**(*simp add*: *nth-hole-def* ‹*n* = *Suc m*›)
qed

Same as previous lemma, but written with a quantifier

lemma *nth-hole-in-forall*: ∀ *n* . *nth-hole n* ∈ *C*
by(*simp add*: *nth-hole-in*)

The assumption made in this context (*is-dissection C*) leads to *False* (since *local.nth-hole* generates an infinite subset of *C*)

theorem *false-if-dissection*: *False*
proof −
have *inj nth-hole* **by**(*rule nth-hole-inj*)
moreover have ∀ *n* . *nth-hole n* ∈ *C* **by**(*rule nth-hole-in-forall*)
— *local.nth-hole* injective, with infinite domain **N**, and image contained in *C* together implies that *C* is infinite
ultimately have *infinite C* **by**(*metis finite-subset image-subset-iff range-inj-infinite*)
moreover have *finite C* **using** *dissection* **unfolding** *is-dissection-def* **by** *metis*
ultimately show *False* **by** *metis*
qed
end — Here we end the *is-dissection C* context

Main result (spelling out the definition of *is-dissection*).

theorem *dissection-does-not-exist*:
∄ *C*. (∀ *c* ∈ *C*. *is-valid c*)
∧ *disjoint* (*image to-set C*)
∧ ⋃(*image to-set C*) = *to-set unit-cube*
∧ *inj-on width C*
∧ *card C* ≥ *2*
∧ *finite C*
by(*metis is-dissection-def false-if-dissection*)
end

References

[1] Formalizing 100 Theorems. https://www.cs.ru.nl/~freek/100/, accessed 2024-11-22.

[2] J. E. Littlewood and B. Bollobas. *Littlewood's Miscellany*. Cambridge University Press, Cambridge [Cambridgeshire] ; New York, rev. ed edition, 1986.

[3] F. van Doorn. Archive.Wiedijk100Theorems.CubingACube. https://leanprover-community.github.io/mathlib4_docs/Archive/Wiedijk100Theorems/CubingACube.html#Theorems100.%C2%AB82%C2%BB.cannot_cube_a_cube, accessed 2024-11-22.