2011.12.19
関根　渓（情報知識ネットワーク研究室　B4）

COMPUTATIONAL GEOMETRY

14. Quadtrees

Non-Uniform Mesh Generation

2011.12.19

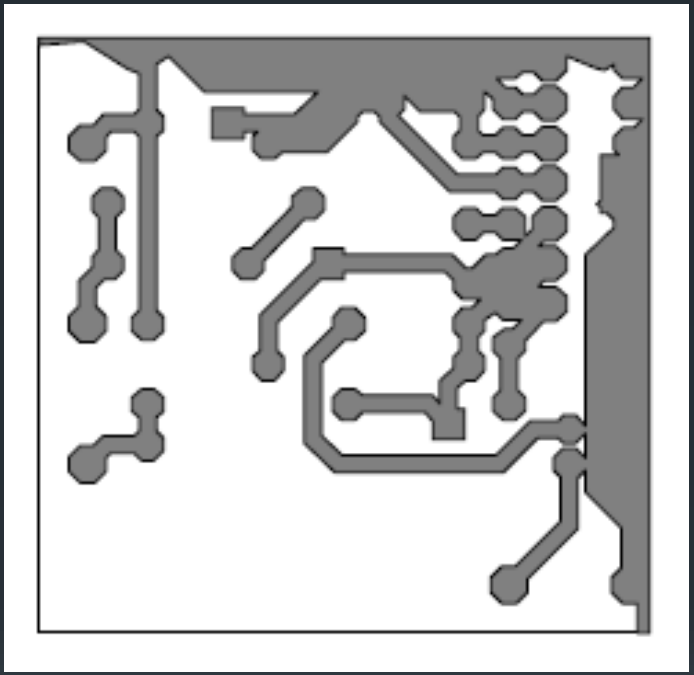
関根　渓（情報知識ネットワーク研究室　B4）

COMPUTATIONAL GEOMETRY

14章 4分木

非一様なメッシュ生成

はじめに・・・

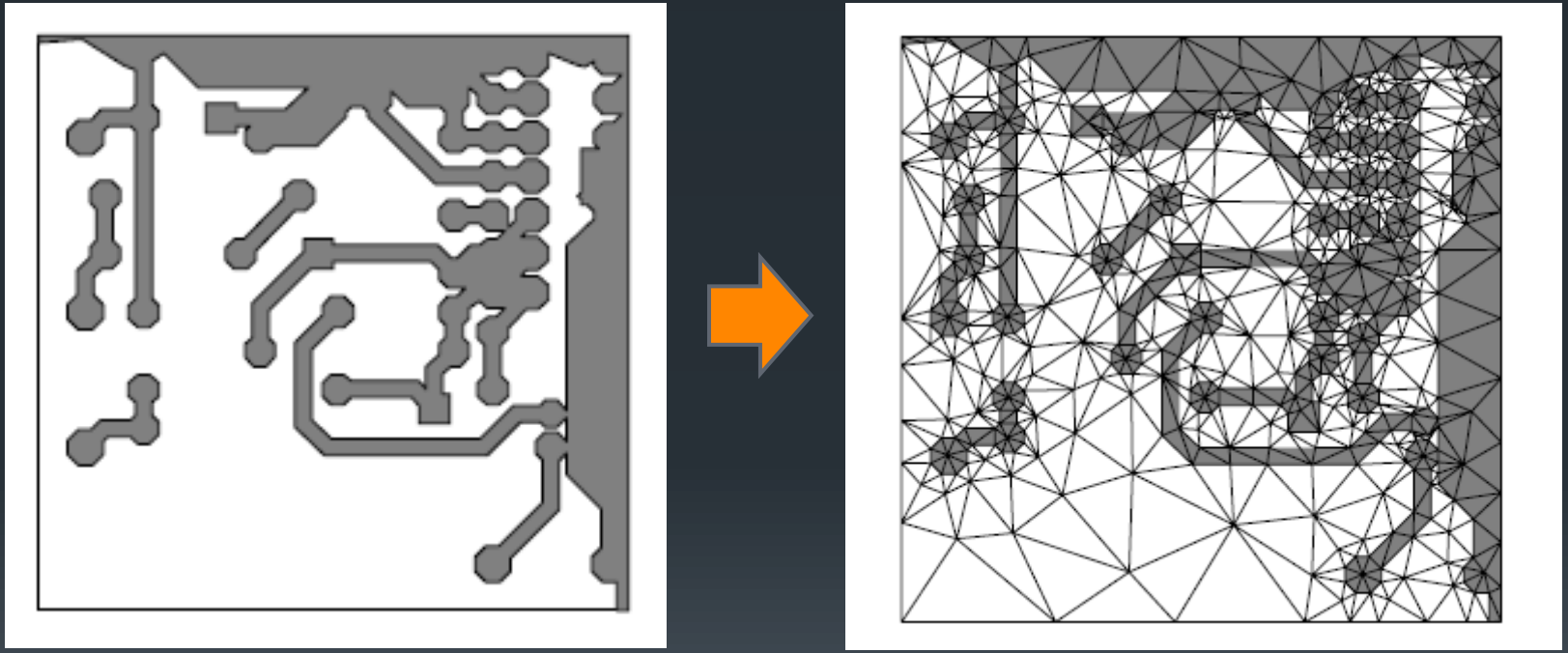


- ほぼ全ての電気機器は機能を制御する電気回路を含んでおり，その回路はプリント回路基板に配置されている
- プリント回路基板を設計するためには，部品を何処に配置し，それらをどのように配線すべきかを決めなければならない

そこで興味深い幾何問題が多数生じるが，本章ではそのうちの1つである
メッシュ生成 (mesh generation)について考える

はじめに・・・

基板が正しく動作するためには，熱放射がある一定以下でなければならない
→基板を多数の小領域（図では三角形）に分割してシミュレーションを行う



このような近似を用いた方法を，**有限要素法(finite element method)**という

はじめに・・・

- メッシュを細かくする

→よい解が得られるが，計算時間が増加

→必要な所でだけ細かなメッシュを使いたい

→この例の場合は異なる材料の領域の境界部分

不規則な形状（非常に細長い三角形等）
の要素を用いると，
数値プロセスの収束に時間がかかることが多い

はじめに・・・

- メッシュを細かくする

→よい解が得られる

適切なメッシュを生成するには，
どうすればいいのか？

不規則な形状（非常に細長い三角形等）
の要素を用いると，
数値プロセスの収束に時間がかかることが多い

CONTENTS

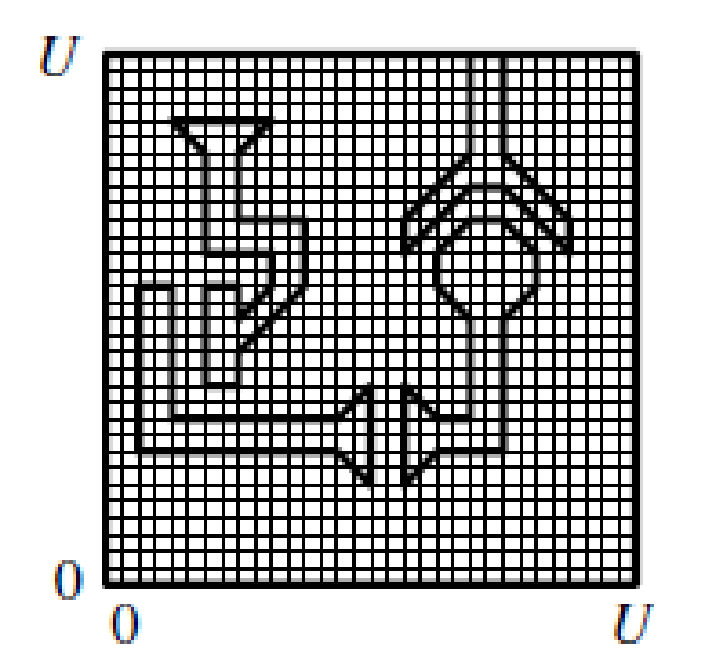
- 14.1 Uniform and Non-Uniform Meshes
  （一様なメッシュと非一様なメッシュ）
- 14.2 Quadtrees for Point Set
  （点集合に対する4分木）
- 14.3 From Quadtrees to Meshes
  （4分木からメッシュへ）

CONTENTS

- 14.1 Uniform and Non-Uniform Meshes
  （一様なメッシュと非一様なメッシュ）
- 14.2 Quadtrees for Point Set
  （点集合に対する4分木）
- 14.3 From Quadtrees to Meshes
  （4分木からメッシュへ）

14.1 一様なメッシュと非一様なメッシュ

- メッシュ生成問題
- 入力：正方形（プリント回路基板）とその中に配置すべき多角形部品
- 出力：正方形の三角形メッシュ(triangular mesh)



- 正方形の頂点は
  $(0,0), (0,U), (U,0), (U,U)$
- ある正定数に対して $U = 2^j$
- 部品の辺の方向は4つに制限
  （辺がx軸となす角度は
  $0°, 45°, 90°, 135°$に限る）

14.1 一様なメッシュと非一様なメッシュ

- メッシュが持たなければいけない性質

- 整合的(comforming)
  - 三角形の辺の中間に別の三角形の頂点があってはならない

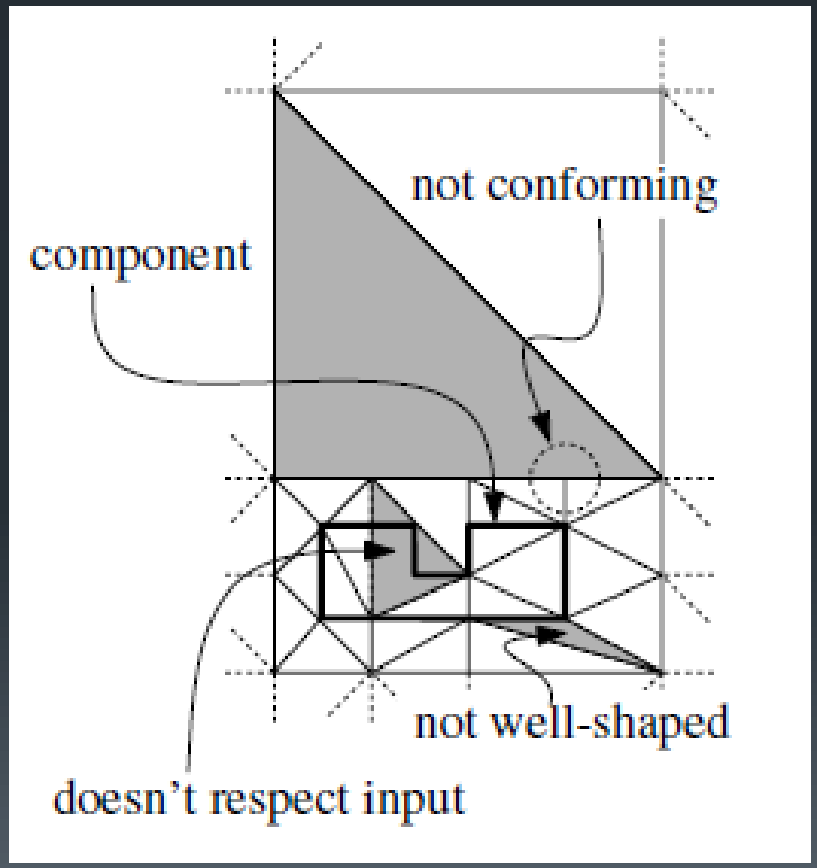
- 入力を尊重(respect the input)
  - 部品の辺はメッシュの三角形の辺の和集合に含まれていなければならない

- よい形(well-shaped)
  - どのメッシュ三角形の角度も $45°$ から $90°$ の間の範囲でなければならない

- 非一様(non-uniform)
  - 部品の辺の近くでは細く，辺から遠い所では荒く

14.1 一様なメッシュと非一様なメッシュ

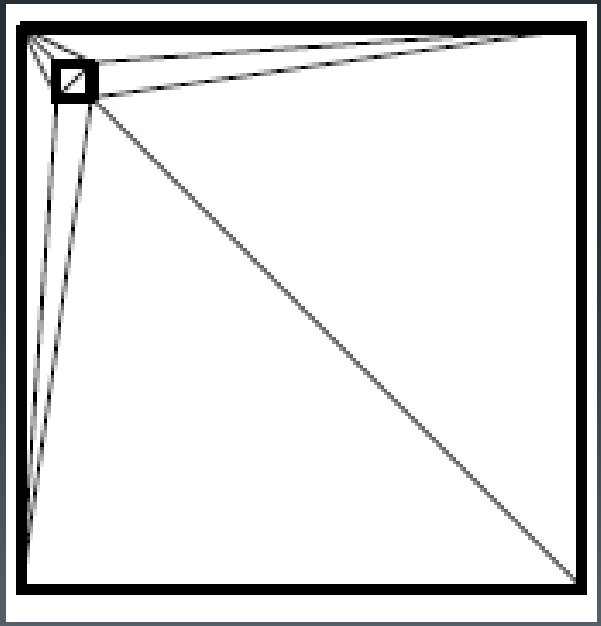
- テキストの例



14.1 一様なメッシュと非一様なメッシュ

- ドロネー三角形分割
  - あらゆる可能な三角形分割の中で最小角度を最大にする分割方式
    →入力を尊重(respect the input)するとは限らない

仮に尊重したとすると・・・？



左図のように,
小さすぎる角度ができてしまうことがあり得る

→三角形が よい形(well-shaped) ではない

14.1 一様なメッシュと非一様なメッシュ

- ドロネー三角形分割
  - ドロネー三角形分割では適切なメッシュ生成は無理なのでは・・？

落とし穴がッ！！

14.1 一様なメッシュと非一様なメッシュ

- ドロネー三角形分割
- メッシュの場合，三角形分割と違って，入力点だけを使って三角形を構成しなければならないという制限はない

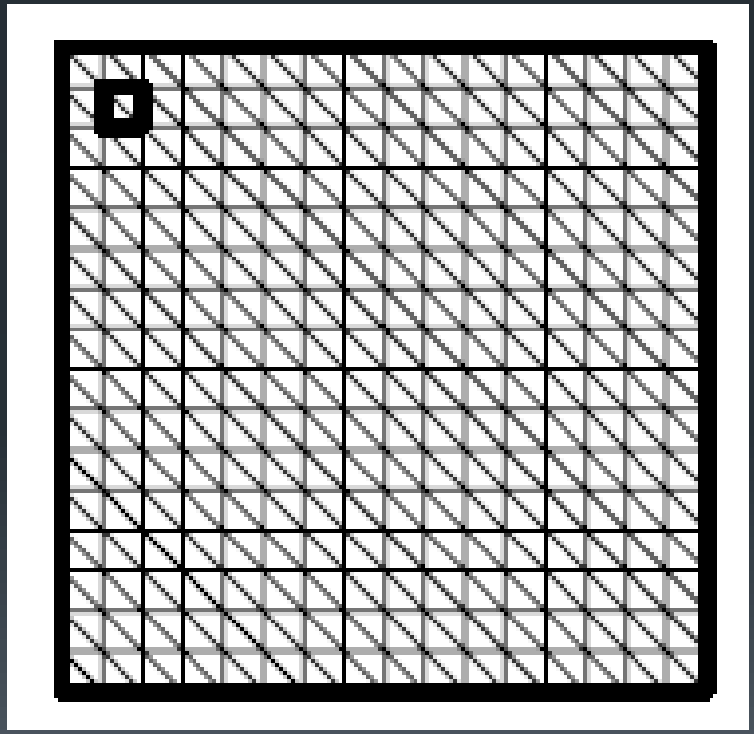
　→よい形(well-shaped) の三角形が得やすいように，点を付け加えて良い

- その余分な点を スタイナ点(Steiner point) と呼ぶ

- スタイナ点を使った三角形分割は，
  スタイナ三角形分割(Steiner triangulation) と呼ばれる

14.1 一様なメッシュと非一様なメッシュ

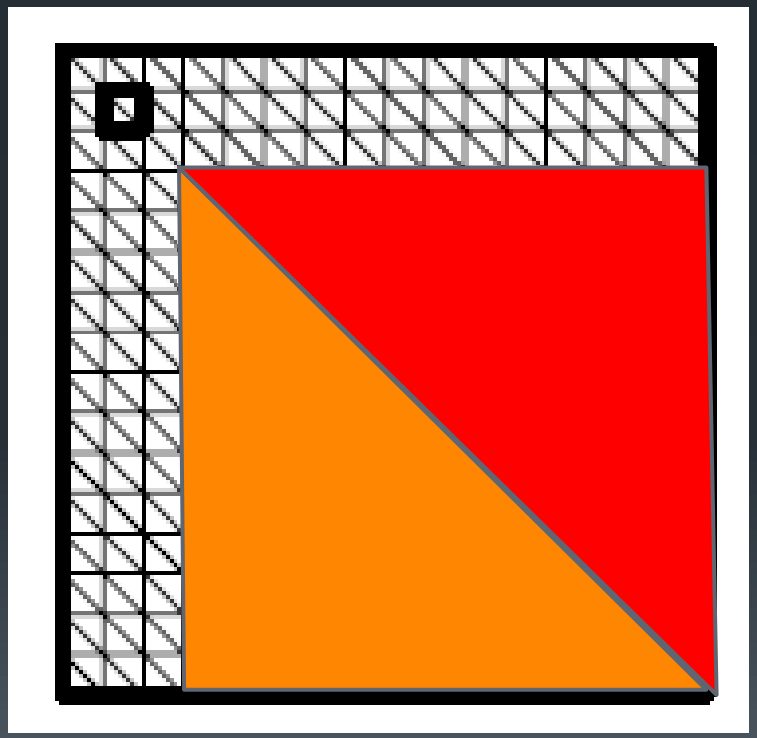
- 正方形内部の全てのグリッド点にスタイナ点を付加
  →45° から 90°の角からなる三角形だけのメッシュが得られた



しかし，これでは一様なメッシュになってしまっている

14.1 一様なメッシュと非一様なメッシュ

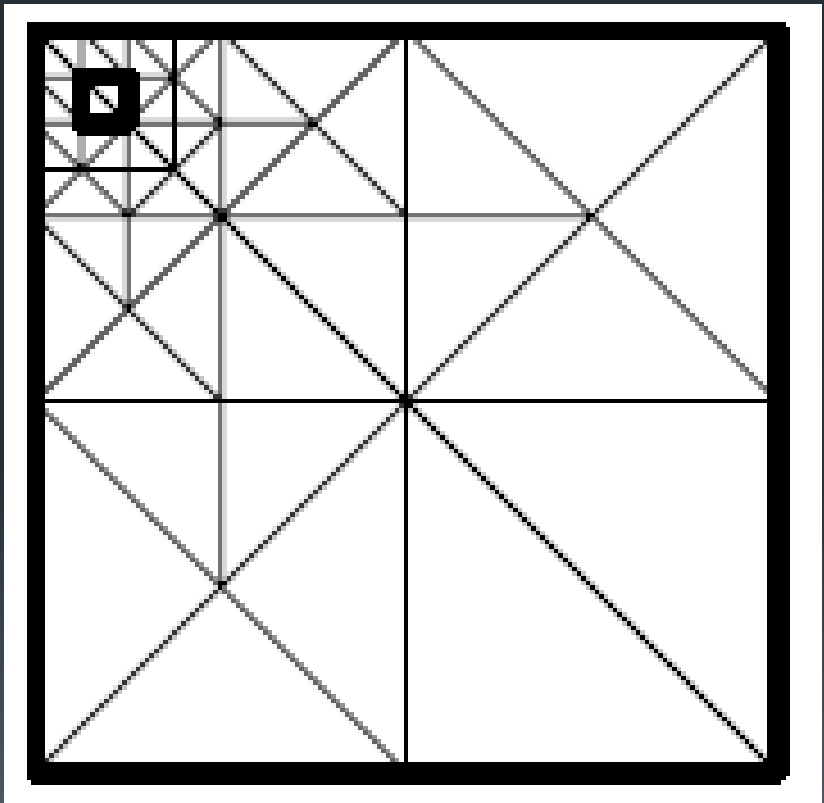
- 右下の部分の三角形を全て2つの大きな三角形で入れ替える
  →整合的(conforming) ではなくなってしまう



ではどうすればいいのか？

14.1 一様なメッシュと非一様なメッシュ

- 左上のコーナーから遠ざかるにつれて三角形を大きくしていく
  →よい形(well-shaped) の三角形だけからなる整合的なメッシュが得られた



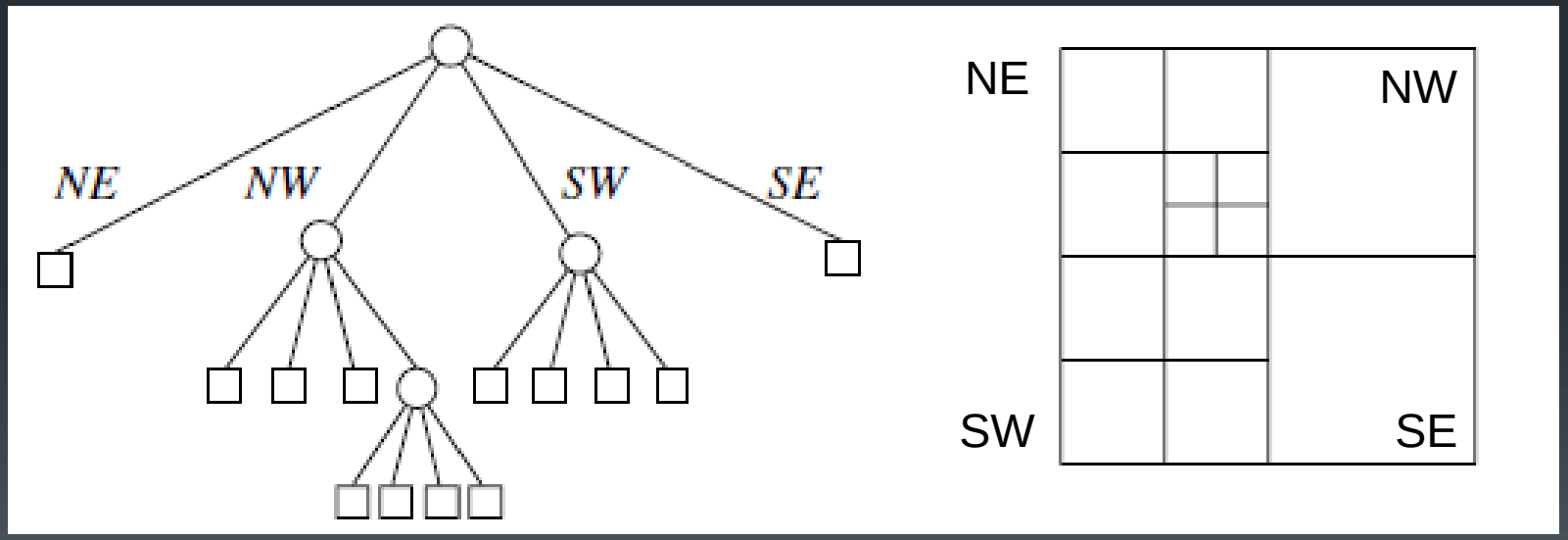
一様なメッシュは512個の三角形を持つが，非一様なメッシュだと52個

CONTENTS

- 14.1 Uniform and Non-Uniform Meshes
  （一様なメッシュと非一様なメッシュ）
- 14.2 Quadtrees for Point Set
  （点集合に対する4分木）
- 14.3 From Quadtrees to Meshes
  （4分木からメッシュへ）

14.2 点集合に対する4分木

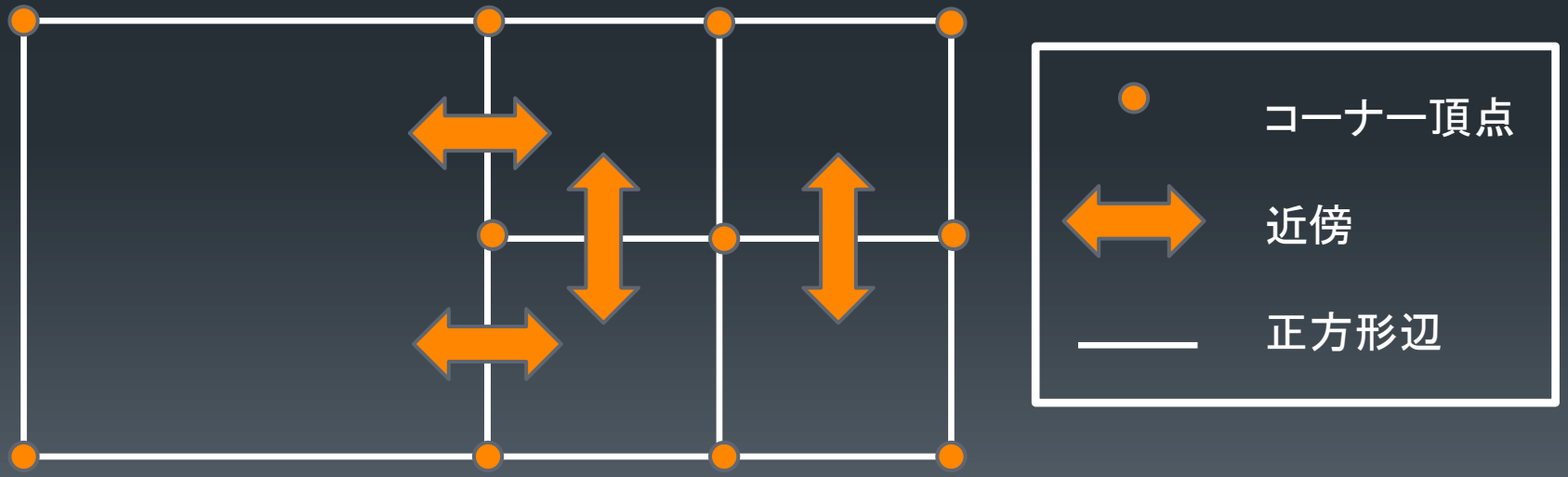
- 4分木(quadtree) とは
  - “全て”の内部節点が4個の子をもつ根付き木
  - どの節点も正方形に対応している
  - 節点vが子をもつとき，その子らに対応する正方形は，vの正方形の4つの象限



根の子のラベル (NE,NW,SW,SE) が対応する象限を表す

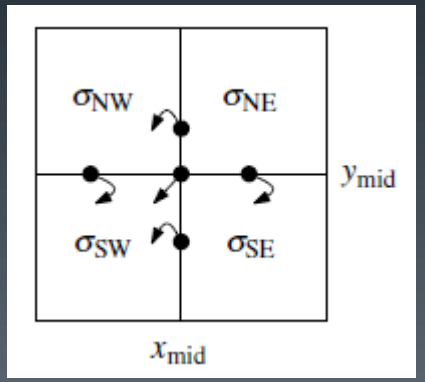
14.2 点集合に対する4分木

- 4分木(quadtree) とは
  - 正方形のコーナーにある4頂点を コーナー頂点(corner vertex), または コーナー(corner) と呼ぶ
  - 正方形の境界に含まれる4分木領域分割の辺を 正方形辺 と呼ぶ
  - 2つの正方形が一辺を共有しているとき，それらは近傍(neighbor) と呼ばれる



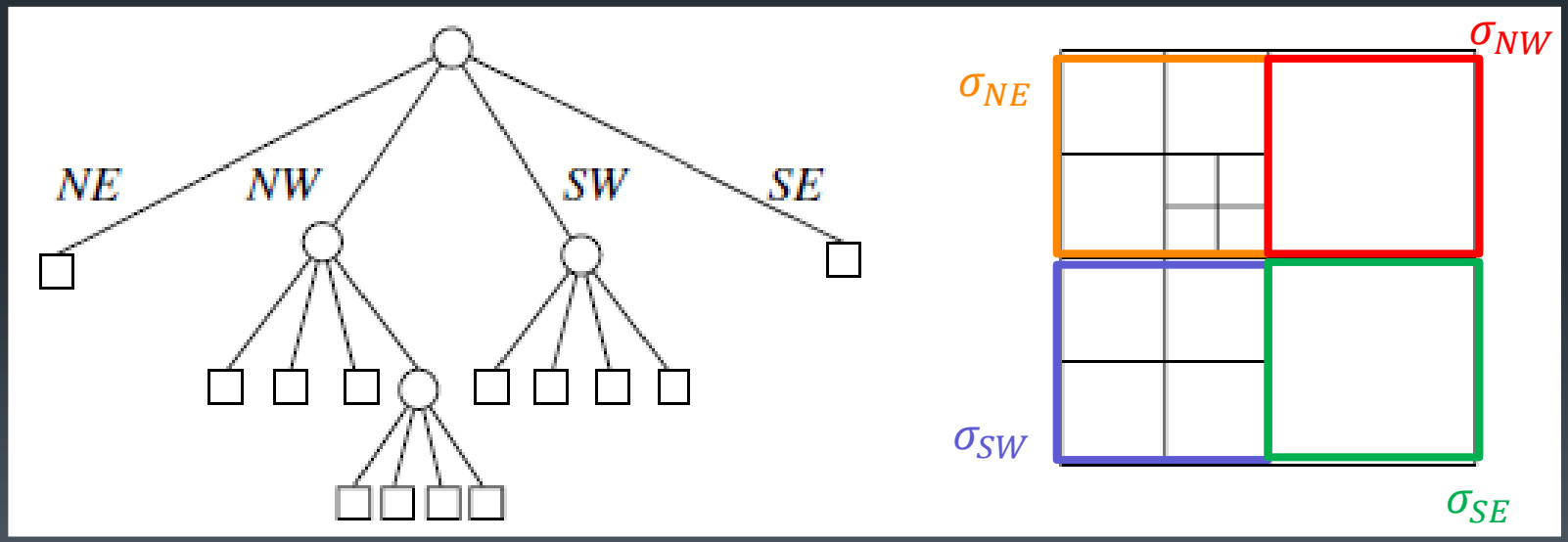
14.2 点集合に対する4分木

- 正方形σの内部の点集合Pに対する4分木の定義
  - $\sigma := [x_\sigma : x'_\sigma] \times [y_\sigma : y'_\sigma]$ とする
  - $card(P) \leq 1$のとき:
    - 4分木は1つの葉節点のみからなり，ここに集合Pと正方形σを蓄える
  - そうでないとき:
    - $x_{mid} := (x_\sigma + x_\sigma')/2$ および $y_{mid} := (y_\sigma + y_\sigma')/2$ として
    - $P_{NE} := \{p \in P : p_x > x_{mid} \ and \ p_y > y_{mid}\}$
    - $P_{NW} := \{p \in P : p_x \leq x_{mid} \ and \ p_y > y_{mid}\}$
    - $P_{SW} := \{p \in P : p_x \leq x_{mid} \ and \ p_y \leq y_{mid}\}$
    - $P_{SE} := \{p \in P : p_x > x_{mid} \ and \ p_y \leq y_{mid}\}$



14.2 点集合に対する4分木

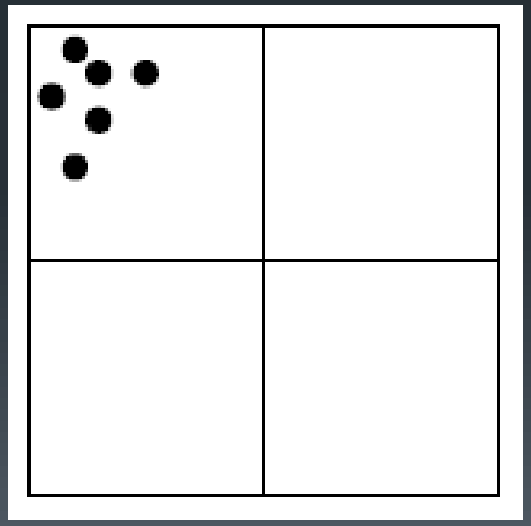
- 正方形σの内部の点集合Pに対する4分木の定義
  - NE-子節点は，正方形σ_{NE}の内部の集合P_{NE}に対する4分木の根
  - NW-子節点は，正方形σ_{NW}の内部の集合P_{NW}に対する4分木の根
  - SW-子節点は，正方形σ_{SW}の内部の集合P_{SE}に対する4分木の根
  - SE-子節点は，正方形σ_{SE}の内部の集合P_{SE}に対する4分木の根



節点vの対応する正方形を$\sigma(v)$とする

14.2 点集合に対する4分木

- 4分木を構成するアルゴリズムの流れ
  1. 現在の正方形を4つの象限に分割
  2. それに従って点集合を分割
  3. 各象限に対して，そこに含まれる点集合に関する4分木を再帰的に構成
  4. 点集合のサイズが1以下になった時に再帰が終了



最初の正方形が入力として与えられないとき
→点集合を包み込む最小の正方形を求める

x, y方向に最も端の点を計算することにより線形時間で実行可能

14.2 点集合に対する4分木

- 4分木の高さ
  - 点を含んでいる正方形は4つの小さな正方形に分割されるが，内部の点集合も分割されるとは限らない

  →4分木は非常に不均衡になり得る

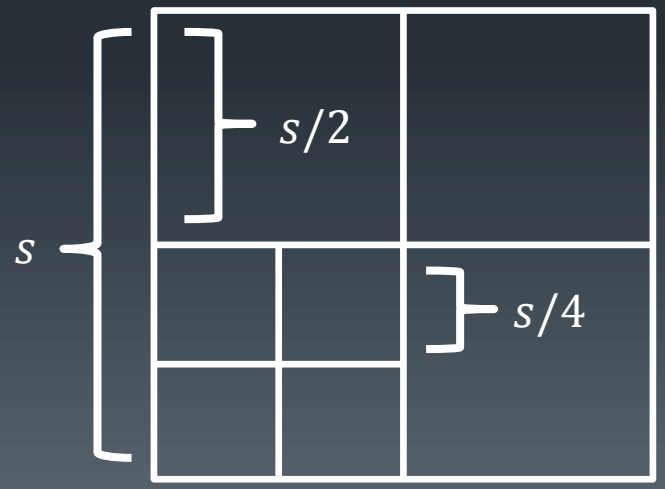
  →4分木のサイズと深さは点の個数の関数では表現不可能

  →4分木の深さは点間の距離と入力正方形のサイズと関係している

14.2 点集合に対する4分木

補題 14.1　平面上の点集合Pに対する4分木の深さは高々 $\log\left(\frac{s}{c}\right) + 3/2$ である．ただし，cはPの任意の2点間の最小距離であり，sはPを含む最初の正方形の一辺の長さである．

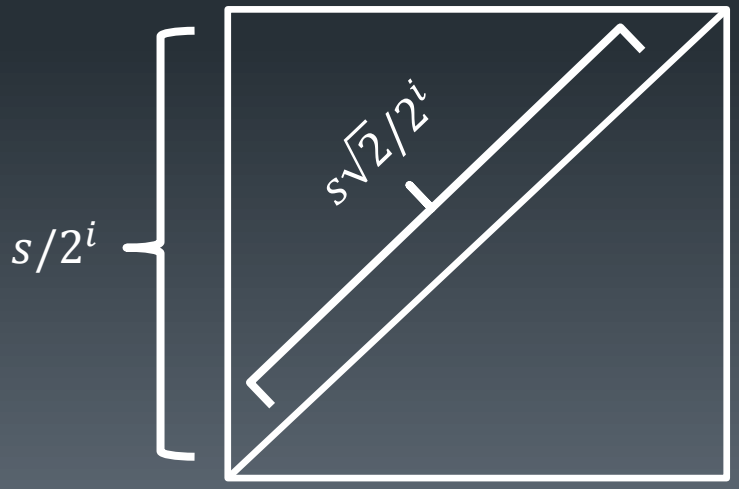
- 証明
  - ある節点からその子節点に降りていくとき，対応する正方形のサイズは半分になる．したがって，深さiの節点の正方形の一辺の長さは $s/2^i$ である．



14.2 点集合に対する4分木

補題 14.1　平面上の点集合Pに対する4分木の深さは高々 $\log\left(\frac{s}{c}\right) + 3/2$ である．ただし，cはPの任意の2点間の最小距離であり，sはPを含む最初の正方形の一辺の長さである．

- 証明
  - 正方形内の2点間の最大距離は，その対角線の長さによって与えられるが，その長さは深さiの節点の正方形に対して$s\sqrt{2}/2^i$である．



14.2 点集合に対する4分木

補題 14.1　平面上の点集合Pに対する4分木の深さは高々 $\log\left(\frac{s}{c}\right)+3/2$ である．ただし，cはPの任意の2点間の最小距離であり，sはPを含む最初の正方形の一辺の長さである．

- 証明
  - 2点間の最小距離はcであるから，内部節点の深さiは次式を満たさなければならない．
    - $s\sqrt{2}/2^{i} \geq \mathrm{c}$

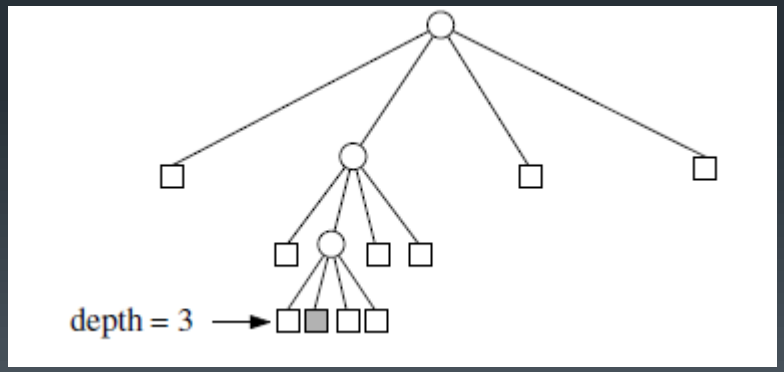
    これから次式を得る

    - $i \leq \log\frac{s\sqrt{2}}{c} = \log\left(\frac{s}{c}\right)+1/2$

14.2 点集合に対する4分木

定理 14.2　n点の集合を蓄える深さdの4分木は，$O((d+1)n)$ 個の節点を持ち，$O((d+1)n)$ 時間で構成することができる.

- 証明
  - 4分木のどの内部節点も4個の子をもつので，葉節点の総数は内部節点数の3倍に1を加えたものである. したがって，内部節点上の上界を求めれば十分である.



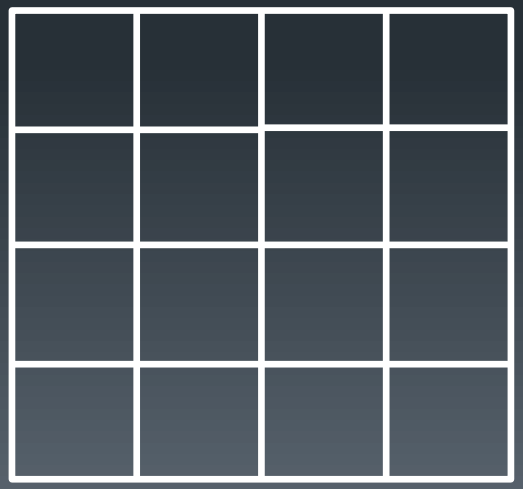
内部節点数 3
葉節点数 10=3*3+1

14.2 点集合に対する4分木

定理 14.2　n点の集合を蓄える深さdの4分木は，$O((d+1)n)$ 個の節点を持ち，$O((d+1)n)$ 時間で構成することができる．

- 証明
  - 任意の内部節点は関連正方形の内部に1点以上を含んでいる．さらに，4分木における同じ深さの節点の正方形は互いに共通部分がなく，最初の正方形をちょうど被覆している．これは任意の与えられた深さにある内部節点の総数は高々nであることを意味している．

深さ=1の節点数は4
その全てが分割されたということは
深さ=1の4つの節点全てに点が含まれる
点数の合計はn
したがって深さ=2の節点数は高々n



14.2 点集合に対する4分木

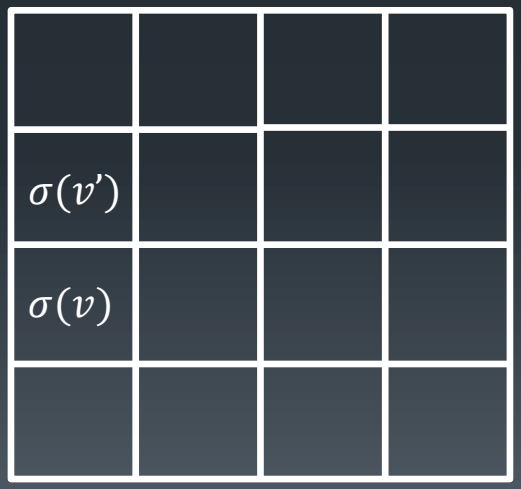
定理 14.2　n点の集合を蓄える深さdの4分木は，$O((d+1)n)$ 個の節点を持ち，$O((d+1)n)$ 時間で構成することができる.

- 証明
  - 再帰的構成アルゴリズムの1つのステップにおける最も時間がかかる仕事は，正方形の象限4つに点を分配することである. したがって，内部節点でかかる時間量は，関連正方形にある点数に関して線形である. 上で木の同じ深さにある節点に関連する点の総数は高々nであることを上で示したが，これから計算量の上界が得られる.（ある高さの正方形に含まれる点数をn，深さ毎に点を分配するのに線形時間かかるということ）

14.2 点集合に対する4分木

- 近傍探索(neighbor finding)
  - 節点vと1つの方向（東西南北のどれか）が与えられたとき，$\sigma(v)$がその方向に隣接している$\sigma(v')$の節点v'を求める操作
  - 4分木領域において与えられた正方形の隣接正方形をみつける

N-近傍の場合



14.2 点集合に対する4分木

- 近傍探索(neighbor finding)
  - N-近傍探索のアルゴリズム

Algorithm NORTHNEIGHBOR(v,T)
Input. A node v in a quadtree T.
Output. The deepest node v whose depth is at most the depth of v such that $o(v)$ is a north-neighbor of $o(v)$, and nil if there is no such node.
1. if v = *root*(T) then return nil
2. if v = SW-child of *parent*(v) then return NW-child of *parent*(v)
3. if v = SE-child of *parent*(v) then return NE-child of *parent*(v)
4. $\mu \leftarrow$NORTHNEIGHBOR(*parent*(v),T)
5. if μ = nil or μ is a leaf
6. then return μ
7. else if v = NW-child of *parent*(v)
8. then return SW-child of μ
9. else return SE-child of μ

14.2 点集合に対する4分木

- 近傍探索(neighbor finding)
  - N-近傍探索のアルゴリズム

入力：4分木Tの節点v

出力：その深さが高々節点vの深さであり，σ(v')，σ(V)のN-近傍であるような最も深い節点v'，およびそのような節点が存在しないときはnil

Algorithm NORTHNEIGHBOR(*v*,T)
Input. A node *v* in a quadtree T.
Output. The deepest node *v* whose depth is at most the depth of *v* such that $\sigma(v)$ is a north-neighbor of $\sigma(v)$, and nil if there is no such node.
1. if *v* = *root*(T) then return nil
2. if *v* = SW-child of *parent*(*v*) then return NW-child of *parent*(*v*)
3. if *v* = SE-child of *parent*(*v*) then return NE-child of *parent*(*v*)
4. $\mu \leftarrow$NORTHNEIGHBOR(*parent*(*v*),T)
5. if μ = nil or μ is a leaf
6. then return μ
7. else if *v* = NW-child of *parent*(*v*)
8. then return SW-child of μ
9. else return SE-child of μ

14.2 点集合に対する4分木

- 近傍探索(neighbor finding)
  - N-近傍探索のアルゴリズム

Algorithm NORTHNEIGHBOR(v,T)
Input. A node v in a quadtree T.
Output. The deepest node v whose depth is at most the depth of v such that $o(v)$ is a north-neighbor of $o(v)$, and nil if there is no such node.
1. if v = *root*(T) then return nil
2. if v = SW-child of *parent*(v) then return NW-child of *parent*(v)
3. if v = SE-child of *parent*(v) then return NE-child of *parent*(v)
4. $\mu \leftarrow$NORTHNEIGHBOR(*parent*(v),T)
5. if μ = nil or μ is a leaf
6. then return μ
7. else if v = NW-child of *parent*(v)
8. then return SW-child of μ
9. else return SE-child of μ

Line1 if文
節点vが根であればnilを返す

Line2 if文
節点がSW-子であればvの親のNW-子を返す

Line3 if文
節点がSE-子であればvの親のNE-子を返す

14.2 点集合に対する4分木

- 近傍探索(neighbor finding)
  - N-近傍探索のアルゴリズム

Algorithm NORTHNEIGHBOR(*v*,T)
Input. A node *v* in a quadtree T.
Output. The deepest node *v* whose depth is at most the depth of *v* such that *o*(*v*) is a north-neighbor of *o*(*v*), and nil if there is no such node.
1. if *v* = *root*(T) then return nil
2. if *v* = SW-child of *parent*(*v*) then return NW-child of *parent*(*v*)
3. if *v* = SE-child of *parent*(*v*) then return NE-child of *parent*(*v*)
4. $\mu \leftarrow$NORTHNEIGHBOR(*parent*(*v*),T)
5. if μ = nil or μ is a leaf
6. then return μ
7. else if *v* = NW-child of *parent*(*v*)
8. then return SW-child of μ
9. else return SE-child of μ

Line4
節点vの親のN-近傍を再帰的に求めて，μに代入

14.2 点集合に対する4分木

- 近傍探索(neighbor finding)
  - N-近傍探索のアルゴリズム

Algorithm NORTHNEIGHBOR(v,T)
Input. A node v in a quadtree T.
Output. The deepest node v whose depth is at most the depth of v such that $o(v)$ is a north-neighbor of $o(v)$, and nil if there is no such node.
1. if v = *root*(T) then return nil
2. if v = SW-child of *parent*(v) then return NW-child of *parent*(v)
3. if v = SE-child of *parent*(v) then return NE-child of *parent*(v)
4. $\mu \leftarrow$NORTHNEIGHBOR(*parent*(v),T)
5. if μ = nil or μ is a leaf
6. then return μ
7. else if v = NW-child of *parent*(v)
8. then return SW-child of μ
9. else return SE-child of μ

Line5-6 if文
μ が nil または葉ならば μ を返す

14.2 点集合に対する4分木

- 近傍探索(neighbor finding)
  - N-近傍探索のアルゴリズム

Algorithm NORTHNEIGHBOR(v,T)
Input. A node v in a quadtree T.
Output. The deepest node v whose depth is at most the depth of v such that $o(v)$ is a north-neighbor of $o(v)$, and nil if there is no such node.
1. if v = *root*(T) then return nil
2. if v = SW-child of *parent*(v) then return NW-child of *parent*(v)
3. if v = SE-child of *parent*(v) then return NE-child of *parent*(v)
4. $\mu \leftarrow$NORTHNEIGHBOR(*parent*(v),T)
5. if μ = nil or μ is a leaf
6. then return μ
7. else if v = NW-child of *parent*(v)
8. then return SW-child of μ
9. else return SE-child of μ

Line7-9 else if文
vがNW-子ならばμのNW-子を返す
そうでないならμのSW-子を返す

14.2 点集合に対する4分木

- 近傍探索(neighbor finding)
  - N-近傍探索のアルゴリズム

Algorithm NORTHNEIGHBOR(v,T)
Input. A node v in a quadtree T.
Output. The deepest node v whose depth is at most the depth of v such that $o(v)$ is a north-neighbor of $o(v)$, and nil if there is no such node.
1. if v = *root*(T) then return nil
2. if v = SW-child of *parent*(v) then return NW-child of *parent*(v)
3. if v = SE-child of *parent*(v) then return NE-child of *parent*(v)
4. $\mu \leftarrow$NORTHNEIGHBOR(*parent*(v),T)
5. if μ = nil or μ is a leaf
6. then return μ
7. else if v = NW-child of *parent*(v)
8. then return SW-child of μ
9. else return SE-child of μ

  - 適当に実演

14.2 点集合に対する4分木

- 近傍探索(neighbor finding)
  - N-近傍探索のアルゴリズム

Algorithm NORTHNEIGHBOR(v,T)
Input. A node v in a quadtree T.
Output. The deepest node v whose depth is at most the depth of v such that $o(v)$ is a north-neighbor of $o(v)$, and nil if there is no such node.
1. if v = *root*(T) then return nil
2. if v = SW-child of *parent*(v) then return NW-child of *parent*(v)
3. if v = SE-child of *parent*(v) then return NE-child of *parent*(v)
4. $\mu \leftarrow$NORTHNEIGHBOR(*parent*(v),T)
5. if μ = nil or μ is a leaf
6. then return μ
7. else if v = NW-child of *parent*(v)
8. then return SW-child of μ
9. else return SE-child of μ

このアルゴリズムは必ずしも葉節点を報告するわけでない
もしそうしたいなら，アルゴリズムで見つけた節点から常に南の子を優先して4分木を降りていけばよい

14.2 点集合に対する4分木

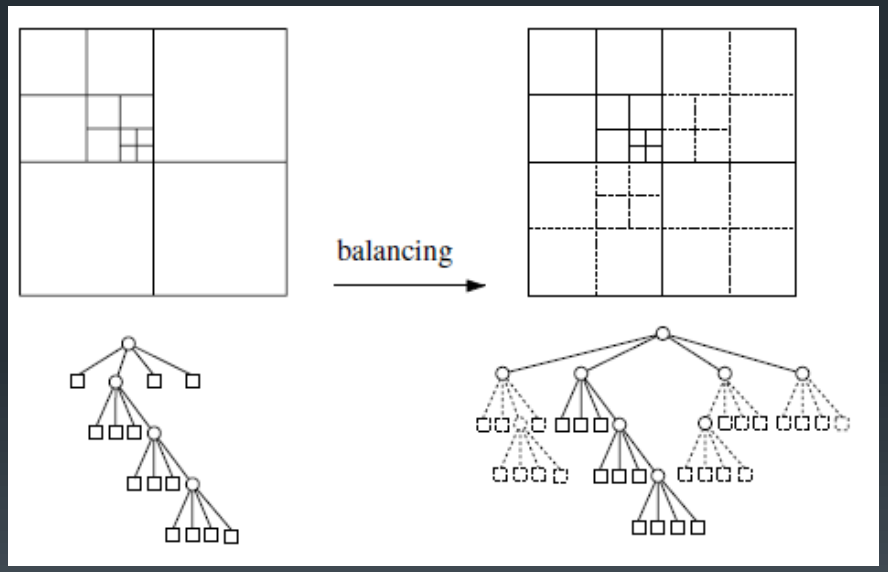
- 近傍探索(neighbor finding)
  - N-近傍探索のアルゴリズム
    - 計算時間の吟味
      - 毎回の呼び出しでアルゴリズムが費やす時間は$O(1)$である
      - 各呼び出しで引数の節点vの深さは1だけ浅くなる

      →実行時間は4分木の深さに対して線形

定理 14.3　Tを深さdの4分木とする．このとき，上で定義したTにおける与えられた節点vの与えられた方向の近傍は $O(d+1)$ 時間で求めることが出来る．

14.2 点集合に対する4分木

- 平衡4分木(balanced quadtree)
  - 均衡がとれている(balanced)とは
    - どの2つの隣接正方形もそのサイズ比が高々2倍
      →平衡4分木ではどの2つの葉節点の高さも高々1しか違わない



空節点を増やして均衡を取るので，本質的には均衡を取るとは言えないが，
メッシュを生成する上では必要な作業

14.2 点集合に対する4分木

- 平衡4分木(balanced quadtree)
  - 4分木の平衡を取るアルゴリズム

Algorithm BALANCEQUADTREE(T)
Input. A quadtree T.
Output. A balanced version of T.
1. Insert all the leaves of T into a linear list L.
2. **while** L is not empty
3. **do** Remove a leaf μ from L.
4. **if** $\sigma(\mu)$ has to be split
5. **then** Make μ into an internal node with four children, which are leaves that correspond to the four quadrants of $\sigma(\mu)$. If μ stores a point, then store the point in the correct new leaf instead.
6. Insert the four new leaves into L.
7. Check if $\sigma(\mu)$ had neighbors that now need to be split and, if so, insert them into L.
8. **return** T

14.2 点集合に対する4分木

- 平衡4分木(balanced quadtree)
  - 4分木の平衡を取るアルゴリズム

入力：4分木T
出力：Tの平衡版

Algorithm BALANCEQUADTREE(T)
Input. A quadtree T.
Output. A balanced version of T.
1. Insert all the leaves of T into a linear list L.
2. **while** L is not empty
3. **do** Remove a leaf μ from L.
4. **if** $\sigma(\mu)$ has to be split
5. **then** Make μ into an internal node with four children, which are leaves that correspond to the four quadrants of $\sigma(\mu)$. If μ stores a point, then store the point in the correct new leaf instead.
6. Insert the four new leaves into L.
7. Check if $\sigma(\mu)$ had neighbors that now need to be split and, if so, insert them into L.
8. **return** T

14.2 点集合に対する4分木

- 平衡4分木(balanced quadtree)
  - 4分木の平衡を取るアルゴリズム

Line1
全ての葉節点を線形リストLに挿入

Algorithm BALANCEQUADTREE(T)
Input. A quadtree T.
Output. A balanced version of T.
1. Insert all the leaves of T into a linear list L.
2. **while** L is not empty
3. **do** Remove a leaf μ from L.
4. **if** $\sigma(\mu)$ has to be split
5. **then** Make μ into an internal node with four children, which are leaves that correspond to the four quadrants of $\sigma(\mu)$. If μ stores a point, then store the point in the correct new leaf instead.
6. Insert the four new leaves into L.
7. Check if $\sigma(\mu)$ had neighbors that now need to be split and, if so, insert them into L.
8. **return** T

14.2 点集合に対する4分木

- 平衡4分木(balanced quadtree)
  - 4分木の平衡を取るアルゴリズム

Algorithm BALANCEQUADTREE(T)
Input. A quadtree T.
Output. A balanced version of T.
1. Insert all the leaves of T into a linear list L.
2. **while** L is not empty
3. **do** Remove a leaf μ from L.
4. **if** $\sigma(\mu)$ has to be split
5. **then** Make μ into an internal node with four children, which are leaves that correspond to the four quadrants of $\sigma(\mu)$. If μ stores a point, then store the point in the correct new leaf instead.
6. Insert the four new leaves into L.
7. Check if $\sigma(\mu)$ had neighbors that now need to be split and, if so, insert them into L.
8. **return** T

Line2-7 whileループ
線形リストLがからになるまで操作を繰り返す

Line3 do文
線形リストLから葉節点μを取り除く

14.2 点集合に対する4分木

- 平衡4分木(balanced quadtree)
  - 4分木の平衡を取るアルゴリズム

Algorithm BALANCEQUADTREE(T)
Input. A quadtree T.
Output. A balanced version of T.
1. Insert all the leaves of T into a linear list L.
2. **while** L is not empty
3. **do** Remove a leaf μ from L.
4. **if** $\sigma(\mu)$ has to be split
5. **then** Make μ into an internal node with four children, which are leaves that correspond to the four quadrants of $\sigma(\mu)$. If μ stores a point, then store the point in the correct new leaf instead.
6. Insert the four new leaves into L.
7. Check if $\sigma(\mu)$ had neighbors that now need to be split and, if so, insert them into L.
8. **return** T

Line4-6 if文
σ(μ)が分割されなくてはいけない正方形ならば，μを分割し，点をその葉節点4つに配分し，その4つの葉節点を線形リストLに挿入

14.2 点集合に対する4分木

- 平衡4分木(balanced quadtree)
  - 4分木の平衡を取るアルゴリズム

Algorithm BALANCEQUADTREE(T)
Input. A quadtree T.
Output. A balanced version of T.
1. Insert all the leaves of T into a linear list L.
2. **while** L is not empty
3. **do** Remove a leaf μ from L.
4. **if** $\sigma(\mu)$ has to be split
5. **then** Make μ into an internal node with four childr ch are leaves that correspond to the four quadrants of $\sigma(\mu)$. stores a point, then store the point in the correct nev af instead.
6. Insert the four new leaves into L.
7. Check if $\sigma(\mu)$ had neighbors that now need to be split and, if so, insert them into L.
8. **return** T

Line7
σ(μ)が分割しなければならない近傍をもっていたかどうかを判定し，もしそうならそれらを線形リストLに挿入する

14.2 点集合に対する4分木

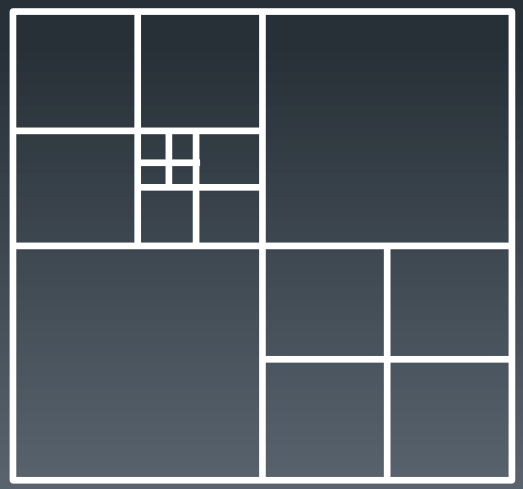
- 平衡4分木(balanced quadtree)
  - 4分木の平衡を取るアルゴリズム
    - $\sigma(\mu)$を分割しなければならないかどうかの判定

      →$\sigma(\mu)$がそのサイズが半分より小さい正方形に隣接しているかどうかを判定すればよい

      →NorthNeighbor(μ,T) が葉節点ではないSW-子，またはSE-子を持つ節点を報告する

    - $\sigma(\mu)$が分割しなければならない近傍をもっていたかどうかの判定

      →NorthNeighbor(μ,T) が対応する正方形が$\sigma(\mu)$より大きい節点を報告する



14.2 点集合に対する4分木

- 平衡4分木(balanced quadtree)
  - 平衡後の4分木のサイズは大きくなりすぎないか？
    - 理由1：非常に小さな正方形に隣接する大きな正方形は何度も分割される
    - 理由2：分割が伝播し，最初は分割対象でなかった正方形が分割され得る

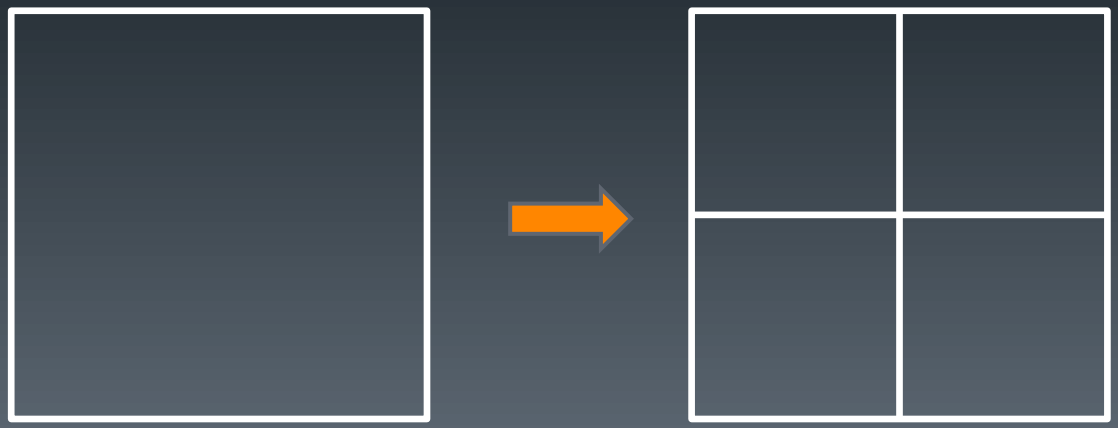
定理 14.4　Tを深さm個の節点をもつ4分木とする．このとき，Tを平衡化したもの$O(m)$個の節点をもち，$O\big((d+1)m\big)$ 時間で構成できる．

14.2 点集合に対する4分木

- 平衡4分木(balanced quadtree)

定理 14.4　Tをm個の節点をもつ4分木とする．このとき，Tを平衡化したものは$O(m)$個の節点をもち，$O\big((d+1)m\big)$ 時間で構成できる．

- 証明
  - 最初に頂点数の上界を証明する．Tを平衡化したものをT_Bと表すことにする．何度も分割の操作を行うことによって，Tから木T_Bが得られるが，各分割で1つの葉節点が4つの葉節点を持つ内部節点に置き換わる．ここで分割の操作が$8m$回しか実行されないことを証明する．分割の操作を1回実行すると，節点（内部および葉）の総数は4だけ増えるので，T_Bの頂点数は主張通り$O(m)$であることを示している．



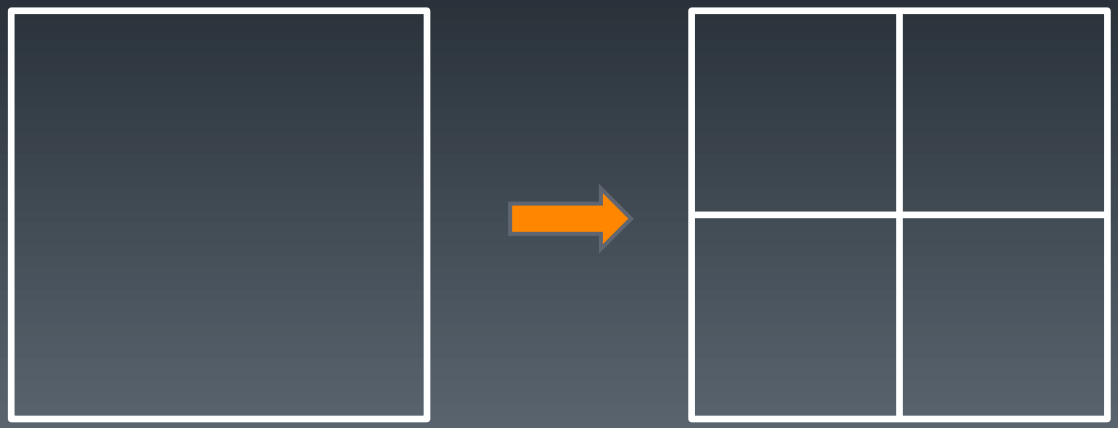
もし分割が$8m$回しか行われないなら，
確かに平衡4分木の節点数は
$$4 * 8m = O(m)$$

14.2 点集合に対する4分木

- 平衡4分木(balanced quadtree)

定理 14.4　Tをm個の節点をもつ4分木とする．このとき，Tを平衡化したものは$O(m)$個の節点をもち，$O\big((d+1)m\big)$ 時間で構成できる．

- 証明
  - 最初に頂点数の上界を証明する．Tを平衡化したものをT_Bと表すことにする．何度も分割の操作を行うことによって，Tから木T_Bが得られるが，各分割で1つの葉節点が4つの葉節点を持つ内部節点に置き換わる．ここで分割の操作が$8m$回しか実行されないことを証明する．分割の操作を1回実行すると，節点（内部および葉）の総数は4だけ増えるので，T_Bの頂点数は主張通り$O(m)$であることを示している．



もし分割が$8m$回しか行われないなら，
確かに平衡4分木の節点数は
$$4 * 8m = O(m)$$

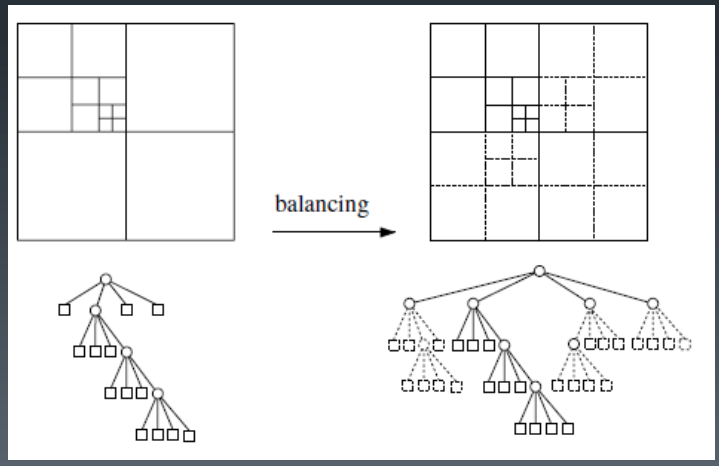
14.2 点集合に対する4分木

- 平衡4分木(balanced quadtree)

定理 14.4　Tをm個の節点をもつ4分木とする．このとき，Tを平衡化したものは$O(m)$個の節点をもち，$O\big((d+1)m\big)$ 時間で構成できる．

- 証明
  - Tの節点の正方形を古い正方形と呼び，T_BにあるがTには含まれない節点の正方形を新しい正方形と呼ぶ．均衡化の過程で正方形σ（新古関係なしに）を分割しなくてはならないとする．（このときσの象限はさらに分割されなければならないかもしれないが，これについては別に考慮する．ここではσの分割によって節点の個数が4だけ増加することについて考えるだけでよい．）

実線が新しい正方形，
点線が新しい正方形

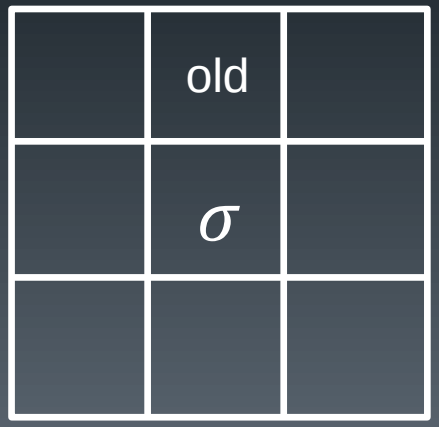


14.2 点集合に対する4分木

- 平衡4分木(balanced quadtree)

定理 14.4　Tをm個の節点をもつ4分木とする．このとき，Tを平衡化したものは$O(m)$個の節点をもち，$O\big((d+1)m\big)$ 時間で構成できる．

- 証明
  - 以下では，σを取り囲む8個の同じサイズの正方形の少なくとも1つは古い正方形でなければならないということを証明する．σの分割をこれらの古い正方形の1つに課金する．古い正方形（Tの全ての節点）はこのようにして高々8回の分割で課金されるので，主張通り，分割の総数は高々$8m$である．



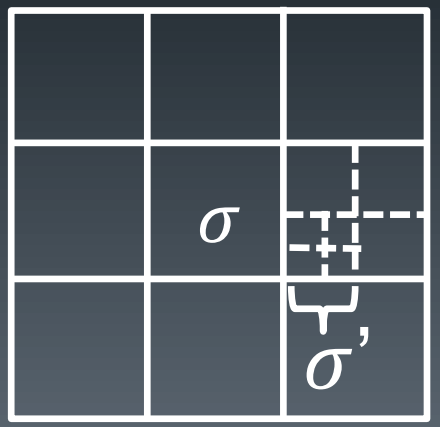
σは周りの古い正方形の子によって，
最大で8回分割され得る，ということ

14.2 点集合に対する4分木

- 平衡4分木(balanced quadtree)

定理 14.4　Tをm個の節点をもつ4分木とする．このとき，Tを平衡化したものは$O(m)$個の節点をもち，$O\big((d+1)m\big)$ 時間で構成できる．

- 証明
  - 均衡化の過程において分割される任意の正方形に対して，それを取り囲む8個の同じサイズの正方形のうち少なくとも1つは古い正方形でなければならないことを証明する．その主張が成り立たないような正方形の中で，最小のものをσとする．仮定よりσは分割されるので，σはその半分のサイズより小さい正方形に隣接されていたはずである．σ’をちょうどσの半分のサイズの正方形で，この小さな正方形を含むものとする．

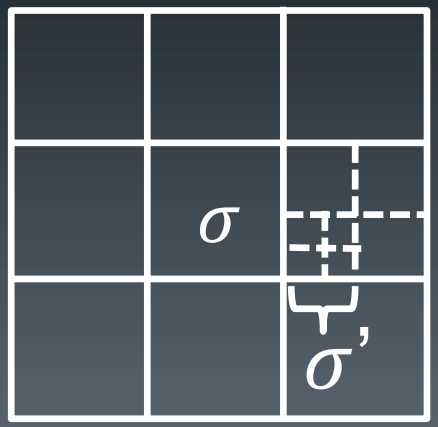


14.2 点集合に対する4分木

- 平衡4分木(balanced quadtree)

定理 14.4　Tをm個の節点をもつ4分木とする．このとき，Tを平衡化したものは$O(m)$個の節点をもち，$O\big((d+1)m\big)$ 時間で構成できる．

- 証明
  - 仮定よりσの周りに古い正方形は存在しない．よってσ'は新しい正方形に含まれているので，それ自体も新しい正方形である．したがって，それは均衡化の仮定において分割されたはずである．ここで，σ'を取り囲む同じサイズの8個の正方形を考えると，

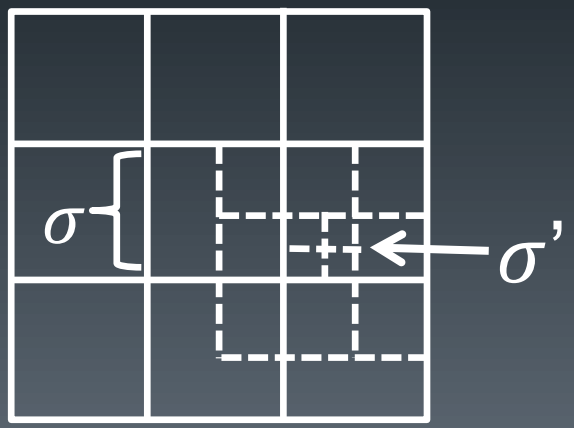


14.2 点集合に対する4分木

- 平衡4分木(balanced quadtree)

定理 14.4　Tをm個の節点をもつ4分木とする．このとき，Tを平衡化したものは$O(m)$個の節点をもち，$O\big((d+1)m\big)$ 時間で構成できる．

- 証明
  - 仮定よりσの周りに古い正方形は存在しない．よってσ'は新しい正方形に含まれているので，それ自体も新しい正方形である．したがって，それは均衡化の仮定において分割されたはずである．ここで，σ'を取り囲む同じサイズの8個の正方形を考えると，それらはσを取り囲む（新しい）正方形の1つに含まれるか，あるいは（均衡化の過程で分割されるものと仮定した）σに含まれているので，それらは新しいものだということがわかる．
  - よって，σ'はσより小さいが，分割される正方形であり，かつそれを取り囲む8個の正方形はどれも古くない．
    これはσの定義に矛盾するので，
    定理の最初の部分の証明を終わる．



14.2 点集合に対する4分木

- 平衡4分木(balanced quadtree)

定理 14.4　Tをm個の節点をもつ4分木とする．このとき，Tを平衡化したものは$O(m)$個の節点をもち，$O\big((d+1)m\big)$ 時間で構成できる．

- 証明
  - 計算時間について
    - 近傍発見の操作が定数回しか実行されない
      →節点 μ を扱うのに必要な時間は $O(d+1)$ （定理14.3 参照）
    - どの節点も高々1回しか扱われず，節点の総数は $O(m)$
      →全体の時間は $O((d+1)m)$

以上で定理14.4の証明は終わり

CONTENTS

- 14.1 Uniform and Non-Uniform Meshes
  （一様なメッシュと非一様なメッシュ）
- 14.2 Quadtrees for Point Set
  （点集合に対する4分木）
- 14.3 From Quadtrees to Meshes
  （4分木からメッシュへ）

14.3 4分木からメッシュへ

- メッシュ生成問題
  - メッシュ生成の第1歩として4分木領域分割を用いる
  - 入力が点集合ではなく多角形なので，停止条件の変更が必要
  - 正方形が1点以下しか含まなかったとき→正方形がどの部品の辺とも交差しなくなるか，単位サイズになったとき

  - 分割が終了した時の正方形

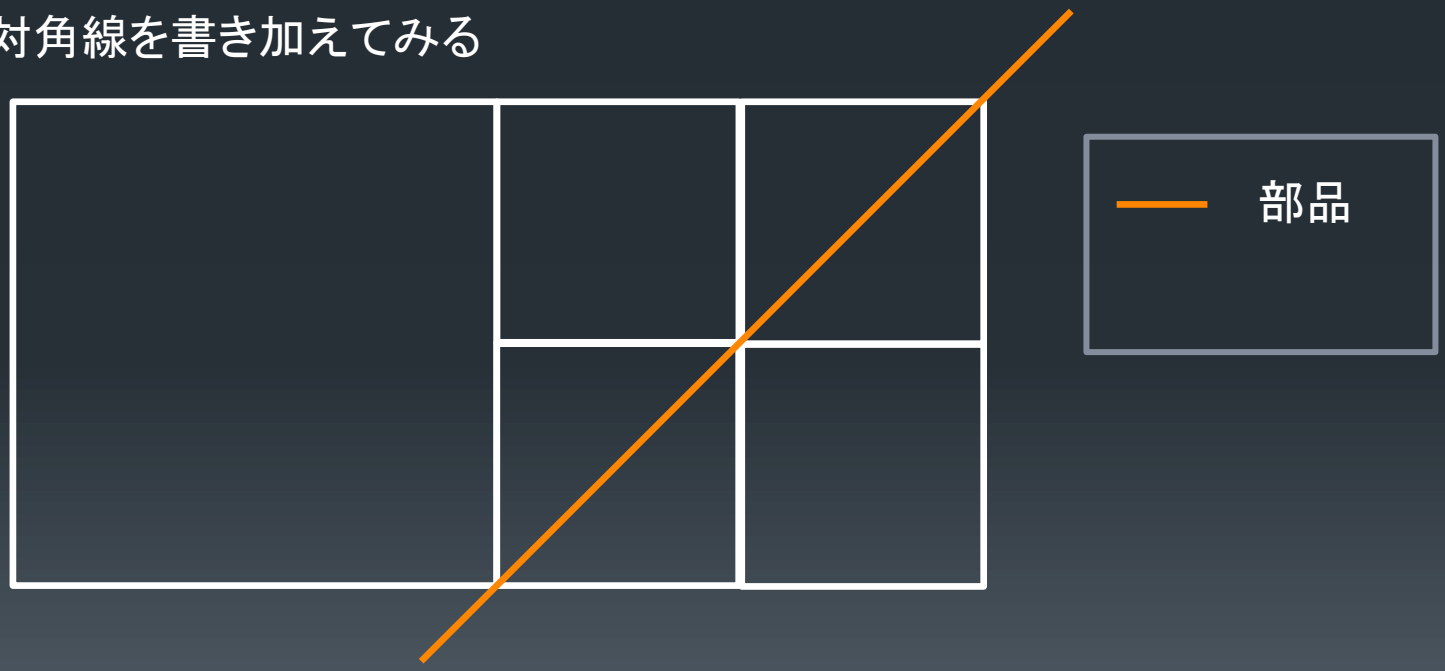
単位サイズかつ対角線で部品の辺と交差するもの

部品の辺と接するか，交わらないもの

※部品の辺は単位サイズの正方形で取り囲まれ，正方形はその辺から遠く離れるほど大きくなるので，4分木領域分割は一様(uniform)にはならない

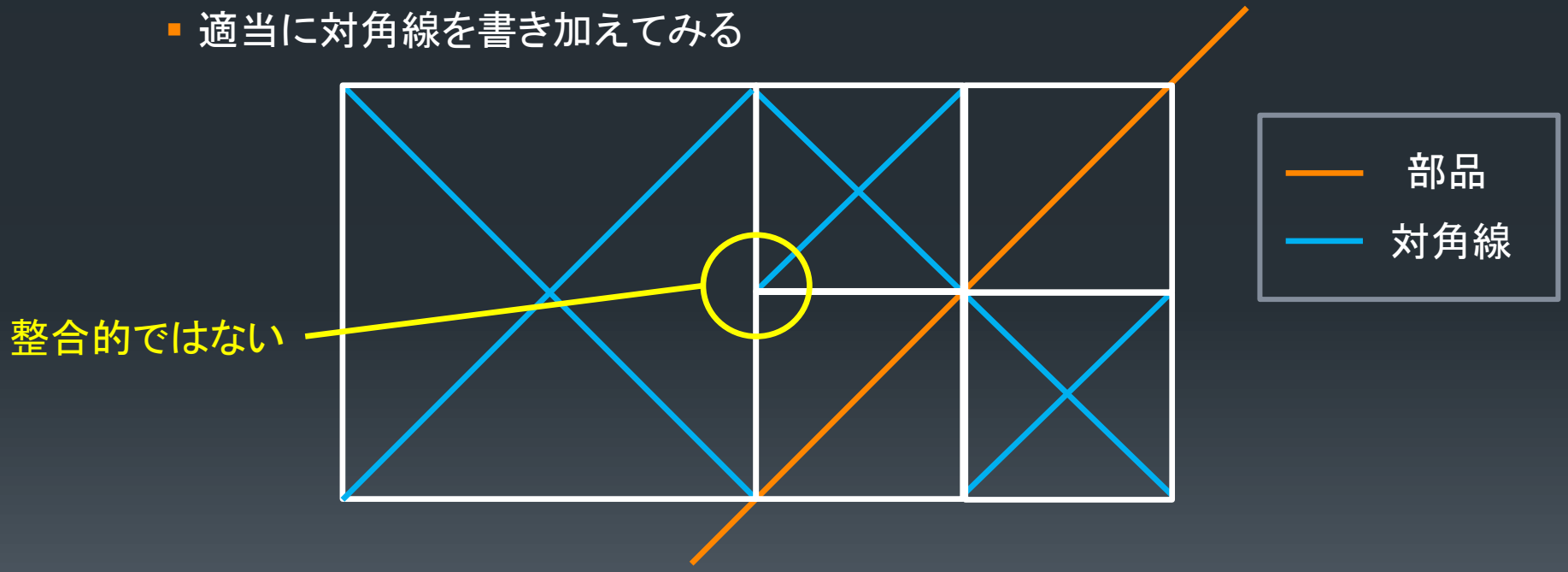
14.3 4分木からメッシュへ

- メッシュ生成問題
  - 良いメッシュを得るためには
    - 部品の辺と交差しない正方形に対角線を加えて三角形にしなくてはいけない
    - 適当に対角線を書き加えてみる



14.3 4分木からメッシュへ

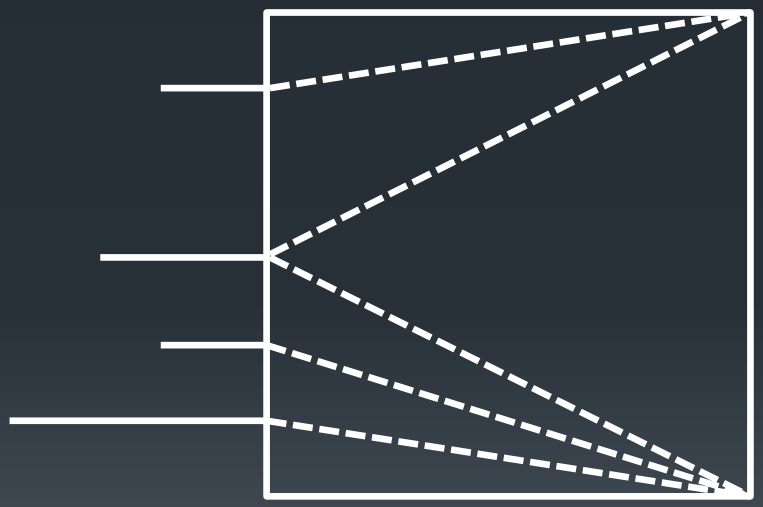
- メッシュ生成問題
  - 良いメッシュを得るためには
    - 部品の辺と交差しない正方形に対角線を加えて三角形にしなくてはいけない
    - 適当に対角線を書き加えてみる



これらのメッシュは入力を尊重しており，形がよく，一様ではないが・・・

14.3 4分木からメッシュへ

- メッシュ生成問題
  - 良いメッシュを得るためには
    - 分割の際には正方形の側辺上の分割頂点を考慮すればよい



正方形がその側辺に多数の頂点を持っていれば，得られる三角形のすべてについて形がよいとは限らない

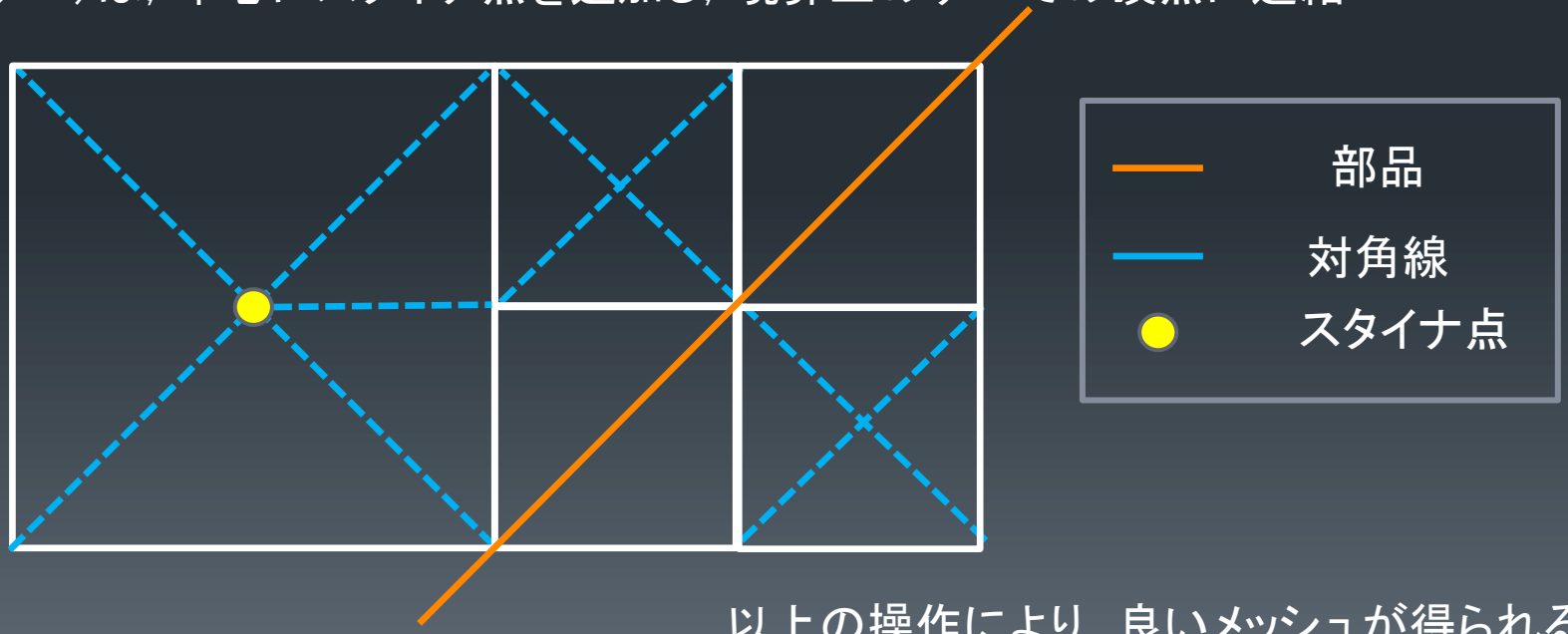
14.3 4分木からメッシュへ

- メッシュ生成問題
  - 良いメッシュを得るためには
    - 解決策

アデルバート・スタイナー from FF9

1. 側辺の内部に頂点を持たない（しかも，成分の辺によって三角形分割されていない）正方形は，対角線により三角形分割

2. その他の正方形（領域分割が平衡しているので，各側辺の頂点は高々一つずつ）は，中心にスタイナ点を追加し，境界上のすべての頂点に連結



以上の操作により，良いメッシュが得られる

14.3 4分木からメッシュへ

- メッシュ生成問題
  - アルゴリズム

Algorithm GENERATEMESH(S)
Input. A set S of components inside the square $[0:U] \times [0:U]$ with the properties stated at the beginning of this section.
Output. A triangular mesh M that is conforming, respects the input, consists of well-shaped triangles, and is non-uniform.

1. Construct a quadtree T on the set S inside the square $[0:U] \times [0:U]$ with the following stopping criterion: a square is split as long as it is larger than unit size and its closure intersects the boundary of some component.
2. $T \leftarrow$ BALANCEQUADTREE(T)
3. Construct the doubly-connected edge list for the quadtree subdivision M corresponding to T.
4. **for** each face σ of M
5. **do if** the interior of σ is intersected by an edge of a component
6. **then** Add the intersection (which is a diagonal) as an edge to M.
7. **else if** σ has only vertices at its corners
8. **then** Add a diagonal of σ as an edge to M.
9. **else** Add a Steiner point in the center of σ, connect it to all vertices on the boundary of σ, and change M accordingly.
10. **return** M

14.3 4分木からメッシュへ

- メッシュ生成問題
  - アルゴリズム

入力：正方形 $[0:U] \times [0:U]$ の内部の部品の集合Sで，本節の最初に述べた性質を持つもの

Algorithm GENERATEMESH(S)
Input. A set S of components inside the square $[0:U] \times [0:U]$ with the properties stated at the beginning of this section.
Output. A triangular mesh M that is conforming, respects the input, consists of well-shaped triangles, and is non-uniform.

1. Construct a quadtree T on the set S inside the square $[0:U] \times [0:U]$ with the following stopping criterion: a square is split as long as it is larger than unit size and its closure intersects the boundary of some component.
2. $T \leftarrow$ BALANCEQUADTREE(T)
3. Construct the doubly-connected edge list for the quadtree corresponding to T.
4. **for** each face σ of M
5. **do if** the interior of σ is intersected by an edge of
6. **then** Add the intersection (which is a diagonal) as an edge to M.
7. **else if** σ has only vertices at its corners
8. **then** Add a diagonal of σ as an edge to M.
9. **else** Add a Steiner point in the center of σ, connect it to all vertices on the boundary of σ, and change M accordingly.
10. **return** M

出力：整合的であり，入力を尊重し，良い形の三角形からなり，一様ではない三角形のメッシュ M

14.3 4分木からメッシュへ

- メッシュ生成問題
  - アルゴリズム

Line1
次の停止条件を用いて，正方形内部にある集合Sに関する4分木Tを構成する:
単位サイズより大きく，その閉包がある部品の境界と交差する限り正方形を分割する

Algorithm GENERATEMESH(S)
Input. A set S of components inside the square $[0 : U] \times [0$ the properties stated at the beginning of this section.
Output. A triangular mesh M that is conforming, respects the input, consists of well-shaped triangles, and is non-uniform.

1. Construct a quadtree T on the set S inside the square $[0 : U] \times [0 : U]$ with the following stopping criterion: a square is split as long as it is larger than unit size and its closure intersects the boundary of some component.
2. $T \leftarrow$ BALANCEQUADTREE(T)
3. Construct the doubly-connected edge list for the quadtree subdivision M corresponding to T.
4. **for** each face σ of M
5. **do if** the interior of σ is intersected by an edge of a component
6. **then** Add the intersection (which is a diagonal) as an edge to M.
7. **else if** σ has only vertices at its corners
8. **then** Add a diagonal of σ as an edge to M.
9. **else** Add a Steiner point in the center of σ, connect it to all vertices on the boundary of σ, and change M accordingly.
10. **return** M

14.3 4分木からメッシュへ

- メッシュ生成問題
  - アルゴリズム

Algorithm GENERATEMESH(S)
Input. A set S of components inside the square $[0 : U$
stated at the beginning of this section.
Output. A triangular mesh M that is conforming, respects [illegible], consists of well-shaped triangles, and is non-uniform.

1. Construct a quadtree T on the set S inside the square $[0 : U] \times [0 : U]$ with the following stopping criterion: a square is split as long as it is larger than unit size and its closure intersects the boundary of some component.
2. $T \leftarrow$ BALANCEQUADTREE(T)
3. Construct the doubly-connected edge list for the quadtree subdivision M corresponding to T.
4. **for** each face σ of M
5. **do if** the interior of σ is intersected by an edge of a component
6. **then** Add the intersection (which is a diagonal) as an edge to M.
7. **else if** σ has only vertices at its corners
8. **then** Add a diagonal of σ as an edge to M.
9. **else** Add a Steiner point in the center of σ, connect it to all vertices on the boundary of σ, and change M accordingly.
10. **return** M

Line2 手続き BALANCEQUADTREE
生成した4分木Tの平衡をとる

14.3 4分木からメッシュへ

- メッシュ生成問題
  - アルゴリズム

Algorithm GENERATEMESH(S)
Input. A set S of components inside the square $[0:U] \times [0:U]$ with the properties stated at the beginning of this section.
Output. A triangular mesh M that is conforming, respects the input, consists of well-shaped triangles, and is non-uniform.

1. Construct a quadtree T on the set S inside the square $[0:U] \times [0:U]$ with the following stopping criterion: a square is split as long as it is larger than unit size and its closure intersects the boundary of some component.
2. $T \leftarrow$ BALANCEQUADTREE(T)
3. Construct the doubly-connected edge list for the quadtree subdivision M corresponding to T.
4. **for** each face σ of M
5. **do if** the interior of σ is intersected by an edge of a
6. **then** Add the intersection (which is a dia
7. **else if** σ has only vertices at its corners
8. **then** Add a diagonal of σ as an edg
9. **else** Add a Steiner point in the cen
   to all vertices on the boundary of σ, and change M accordingly.
10. **return** M

Line3
Tに対応する4分木領域分割Mに対する2重連結辺リストを構成する

14.3 4分木からメッシュへ

- メッシュ生成問題
  - アルゴリズム

Algorithm GENERATEMESH(S)
Input. A set S of components inside the square $[0:U] \times [0:U]$ with the properties stated at the beginning of this section.
Output. A triangular mesh M that is conforming, respects the input, consists of well-shaped triangles, and is non-uniform.

1. Construct a quadtree T on the set S inside the square $[0:U] \times [0:U]$ with the following stopping criterion: a square is split as long as it is longer than unit size and its closure intersects the boundary o
2. $T \leftarrow$ BALANCEQUADTREE(T)
3. Construct the doubly-connected edge list fo corresponding to T.
4. **for** each face σ of M
5. **do if** the interior of σ is intersected by an edge of a component
6. **then** Add the intersection (which is a diagonal) as an edge to M.
7. **else if** σ has only vertices at its corners
8. **then** Add a diagonal of σ as an edge to M.
9. **else** Add a Steiner point in the center of σ, connect it to all vertices on the boundary of σ, and change M accordingly.
10. **return** M

Line4-9 for 文
二重連結リストに含まれる全ての正方形σに対して以下の処理を行う

14.3 4分木からメッシュへ

- メッシュ生成問題
  - アルゴリズム

Algorithm GENERATEMESH(S)
Input. A set S of components inside the square [0 :U
stated at the beginning of this section.
Output. A triangular mesh M that is conforming, re
well-shaped triangles, and is non-uniform.
1. Construct a quadtree T on the set S inside the square ... with the following stopping criterion: a square is split ... it is larger than unit size and its closure intersects the bound... some component.
2. $T \leftarrow$ BALANCEQUADTREE(T)
3. Construct the doubly-connected ed... list for the quadtree subdivision M corresponding to T.
4. **for** each face σ of M
5. **do if** the interior of σ is intersected by an edge of a component
6. **then** Add the intersection (which is a diagonal) as an edge to M.
7. **else if** σ has only vertices at its corners
8. **then** Add a diagonal of σ as an edge to M.
9. **else** Add a Steiner point in the center of σ, connect it to all vertices on the boundary of σ, and change M accordingly.
10. **return** M

Line5-6 if 文
正方形σの内部がある部品の辺と交差するならば，Mにその対角線を辺として加える

14.3 4分木からメッシュへ

- メッシュ生成問題
  - アルゴリズム

Algorithm GENERATEMESH(S)
Input. A set S of components inside the square [0 :U
stated at the beginning of this section.
Output. A triangular mesh M that is conforming, re
well-shaped triangles, and is non-uniform.
1. Construct a quadtree T on the set S inside the square [ ] with the following stopping criterion: a square is split as lo larger than unit size and its closure intersects the boundary of so onent.
2. $T \leftarrow$ BALANCEQUADTREE(T)
3. Construct the doubly-connected edge list for the qua subdivision M corresponding to T.
4. **for** each face σ of M
5. **do if** the interior of σ is intersected by an edge o component
6. **then** Add the intersection (which is a diag nal) as an edge to M.
7. **else if** σ has only vertices at its corners
8. **then** Add a diagonal of σ as an edge to M.
9. **else** Add a Steiner point in the center of σ, connect it to all vertices on the boundary of σ, and change M accordingly.
10. **return** M

Line7-8 else if 文
σがコーナーにしか頂点を持たない正方形の場合，その対角線をMの辺として加える

14.3 4分木からメッシュへ

- メッシュ生成問題
  - アルゴリズム

Algorithm GENERATEMESH(S)
Input. A set S of components inside the squa
stated at the beginning of this section.
Output. A triangular mesh M that is conform
well-shaped triangles, and is non-uniform.

1. Construct a quadtree T on the set S inside the sq] × [0 :U] with the following stopping criterion: a square is split s it is larger than unit size and its closure intersects the boundary o omponent.
2. $T \leftarrow$ BALANCEQUADTREE(T)
3. Construct the doubly-connected edge list for the qua subdivision M corresponding to T.
4. **for** each face σ of M
5. **do if** the interior of σ is intersected by an edge of a co onent
6. **then** Add the intersection (which is a diagonal) as n edge to M.
7. **else if** σ has only vertices at its corners
8. **then** Add a diagonal of σ as an edge to M.
9. **else** Add a Steiner point in the center of σ, connect it to all vertices on the boundary of σ, and change M accordingly.
10. **return** M

Line9 else 文
正方形σの中心にスタイナ点を加え，それをσの境界上の全ての頂点に連結し，それにしたがってMを変更する

14.3 4分木からメッシュへ

- メッシュ生成問題

定理 14.5　Sを本節の最初に述べた性質をもつ正方形 $[0:U]\times[0:U]$ の内部にある互いに共通部分をもたない多角形成分の集合とする．このとき，この入力に対する非一様なメッシュで，入力に関して整合的であり，入力を尊重し，かつ形のよい三角形だけを持つものがある．三角形の個数は $O(p(S)log^2U)$ である．ただし $p(S)$ はSの成分の周囲長の和である．また，このメッシュは $O(p(S)log^2U)$ の時間で構成できる．

- 証明
  - メッシュが非一様で，整合的であり，入力を尊重するものであり，しかも形のよい三角形のみからなるという性質は，以上の議論から得ることができる．
  - 示さなければいけないのは，メッシュのサイズと処理時間の上界である．

14.3 4分木からメッシュへ

- メッシュ生成問題

定理 14.5　Sを本節の最初に述べた性質をもつ正方形 $[0:U]\times[0:U]$ の内部にある互いに共通部分をもたない多角形成分の集合とする．このとき，この入力に対する非一様なメッシュで，入力に関して整合的であり，入力を尊重し，かつ形のよい三角形だけを持つものがある．三角形の個数は $O(p(S)log^2U)$ である．ただし $p(S)$ はSの成分の周囲長の和である．また，このメッシュは $O(p(S)log^2U)$ の時間で構成できる．

- 証明
  - メッシュが非一様で，整合的であり，入力を尊重するものであり，しかも形のよい三角形のみからなるという性質は，以上の議論から得ることができる．
  - 示さなければいけないのは，メッシュのサイズと処理時間の上界である．

14.3 4分木からメッシュへ

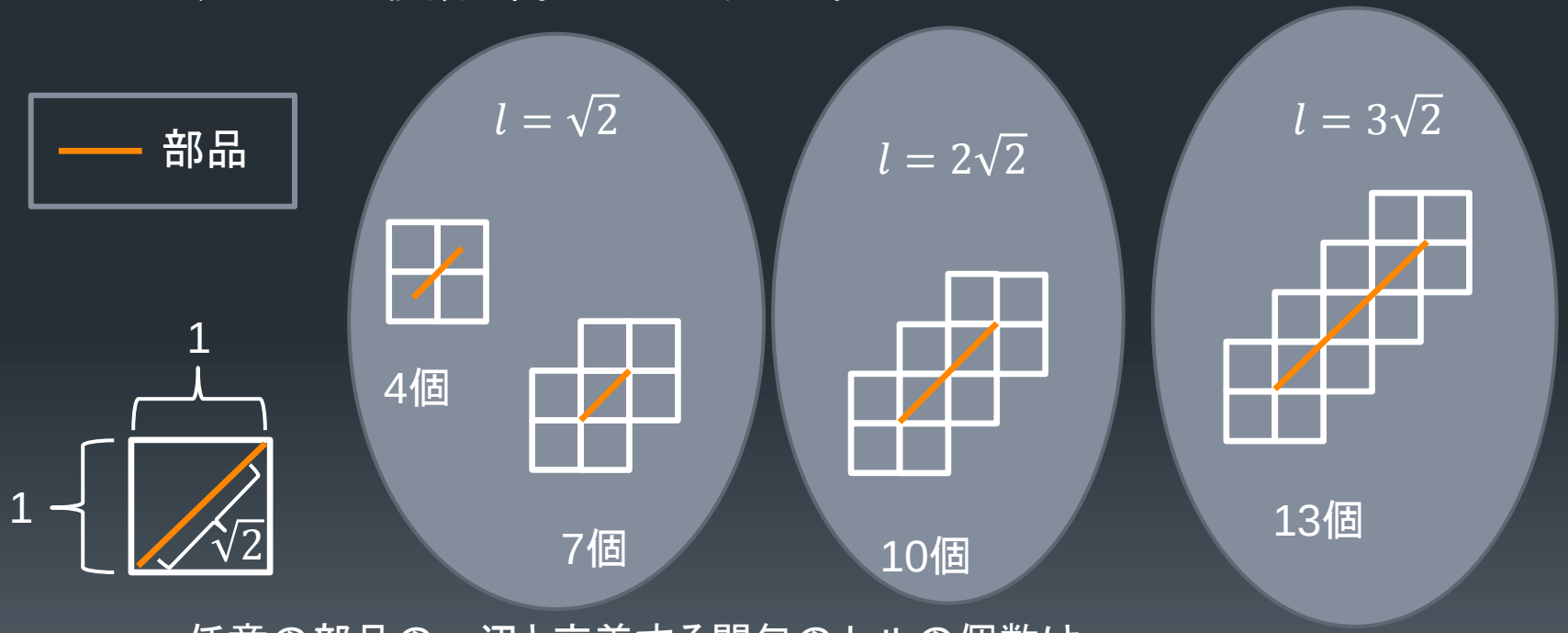
- メッシュ生成問題

定理 14.5　Sを本節の最初に述べた性質をもつ正方形 $[0:U]\times[0:U]$ の内部にある互いに共通部分をもたない多角形成分の集合とする．このとき，この入力に対する非一様なメッシュで，入力に関して整合的であり，入力を尊重し，かつ形のよい三角形だけを持つものがある．三角形の個数は $O(p(S)log^2U)$ である．ただし $p(S)$ はSの成分の周囲長の和である．また，このメッシュは $O(p(S)log^2U)$ の時間で構成できる．

- 証明
  - メッシュの構成は3つの段階に分けて行われる．
    1. 4分木領域分割を構成
    2. 生成された4分木を平衡化する
    3. 三角形分割を行う
  - まずは第一段階で得られる4分木領域分割のサイズの上界を求める．

14.3 4分木からメッシュへ

- メッシュ生成問題
  - 定理14.5(後半部分)証明
    - 1. 第一段階で得られる4分木領域分割のサイズの上界を求める.
    - 単位サイズのセルを持つグリッドを考える. その閉包がその部品のどれかの辺と交差するセルの個数は高々 $4 + 3l\sqrt{2}$ である.



  - 任意の部品の一辺と交差する閉包のセルの個数は
    $O(4 + 3l\sqrt{2}) = O(l) = O(p(S))$

14.3 4分木からメッシュへ

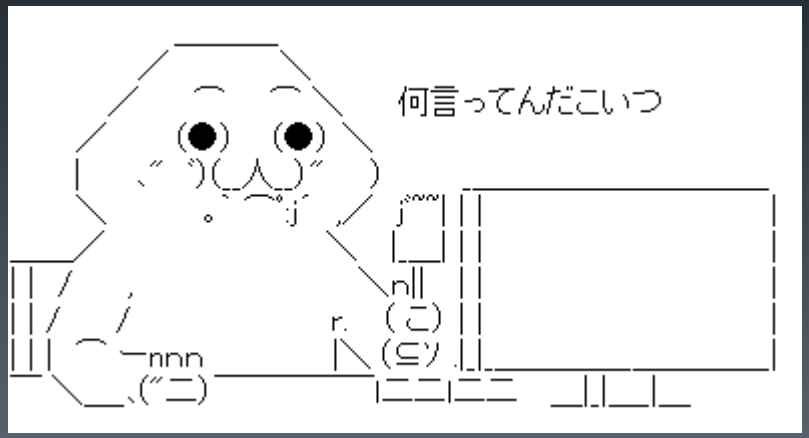
- メッシュ生成問題
  - 定理14.5（後半部分）証明
    - 1. 第一段階で得られる4分木領域分割のサイズ（複雑度）の上界を求める.
    - 任意の部品の一辺と交差する閉包のセルの個数は

$$O\left(4+3l\sqrt{2}\right)=O(l)=O\left(p(S)\right)$$

→明らかにより大きなセルを持つグリッドに対しても成り立つ

→ある固定の深さにある4分木の内部節点の個数は $O(p(S))$

（部品の辺と交わるセルの個数の上界で，内部接点の個数の上界を決定できるのはなぜ・・？）



この謎が解決したとして・・・

4分木の深さは $O(logU)$
（分割は単位サイズになった時終了する）
なので，
分割の複雑度は $O(p(S)logU)$

14.3 4分木からメッシュへ

- メッシュ生成問題
  - 定理14.5（後半部分）証明
    - 2. 第二段階で得られる平衡4分木のサイズの上界を求める.
      →定理14.4より，平衡化にではその複雑度は変化しない
      →平衡4分木の複雑度も $O(p(S)logU)$
    - 3. 三角形分割されて生成されたメッシュのサイズの上界を求める.
      →一つの正方形あたり，最大で8分割しかされない
      →メッシュの複雑度も $O(p(S)logU)$

定理 14.5　Sを本節の最初に述べた性質をもつ正方形 $[0:U]\times[0:U]$ の内部にある互いに共通部分をもたない多角形成分の集合とする．このとき，この入力に対する非一様なメッシュで，入力に関して整合的であり，入力を尊重し，かつ形のよい三角形だけを持つものがある．三角形の個数は $O(p(S)log^2U)$ である．ただし $p(S)$ はSの成分の周囲長の和である．また，このメッシュは $O(p(S)log^2U)$ の時間で構成できる．

14.3 4分木からメッシュへ

- メッシュ生成問題
  - 定理14.5（後半部分）証明
    - 1. 第一段階の4分木分割構成にかかる時間
      - ある深さにおいて，部品の辺と交差する正方形が存在すればそれを分割する．
      - ある深さにおいて，部品の辺と交わる正方形は $O(p(S))$ 個存在する．
      - 分割には深さあたり $O(p(S))$ 時間かかる．
      - 4分木の深さは $O(logU)$ なので，4分木分割の構成には $O(p(S)logU)$ かかる
    - 2. 第二段階の4分木平衡化にかかる時間
      - 定理14.4より，$O(m)$個の節点をもつ4分木の平衡4分木は $O((d+1)m)$ 時間で構成できる（dは4分木の高さ）．
      - →メッシュに用いる4分木お節点数は $O(p(S)logU)$，深さは $O(logU)$ なので，これの平衡化には $O(p(S)log^2U)$ 時間かかる．
    - 3. 第三段階の三角形分割にかかる時間
      - 三角形の数は正方形の数に線形のため，三角形分割には $O(p(S)logU)$ かかる．
    - 二重連結辺リストの構成は三角形分割と同じ時間でできる（らしい）．

以上のことから，メッシュ生成にかかる時間は全体で $O(p(S)log^2U)$ である