Chapter 2
Motion Planning

2.1 Introduction

2.2 Goal of motion planning, Basic problem and extension and Examples of applications

2.3 Brief review of Euclidean Geometry

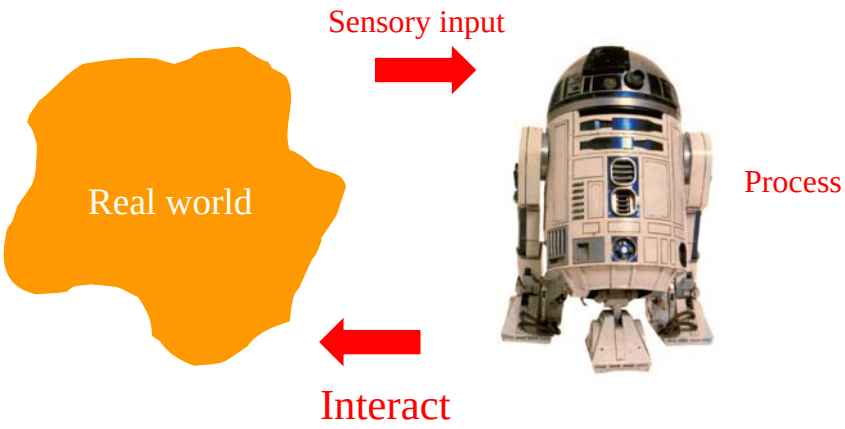
2.4 Basic motion planner

Introduction

• **Robots**	• **Computers**
Manipulate physical objects in the real world under real physical constraint.	Manipulate data in the virtual world under turing computation.

โปรแกรมหุ่นให้ interact กับ real world

เพื่อให้หุ่นทำงานตามที่เราต้องการ



AI:
study of computation that make computers…

Intelligently Perceive, Reason and React.

Marvin Minsky (บิดาแห่ง AI) กล่าวไว้ว่า *"AI is the science of making machines do things that would require intelligence if done by men"*

What is intelligence?

Marvin Minsky now believes the field has taken a wrong turn. Rather than trying to build costly, clumsy physical robots, he says, researchers should concentrate their efforts entirely on computer simulations – that's the key to unraveling the nature of intelligence.

Manipulate an Object

1. เคลื่อนหุ่นให้เข้าใกล้วัตถุเพียงพอที่จะดำเนินการต่อในขั้นตอนต่อไป
2. หยิบ จับ ผลัก หรือทำอย่างไรก็ตามให้วัตถุไปอยู่ในลักษณะที่ต้องการ

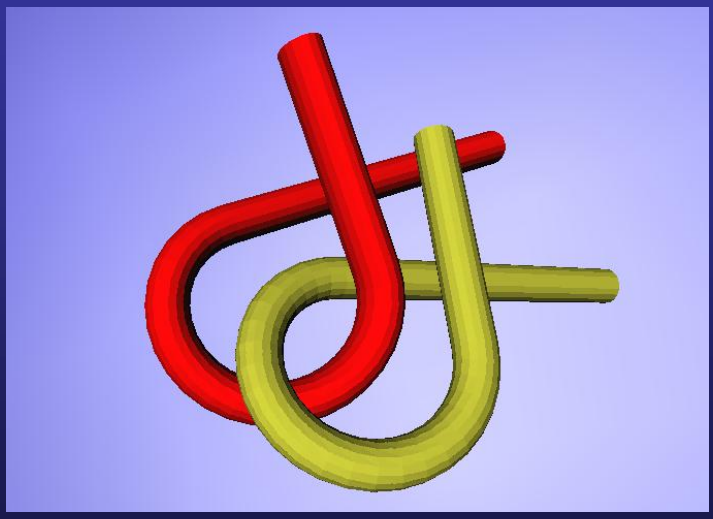
ในบทนี้นี้เราจะพิจารณาขั้นตอนที่ 1

ซึ่งรู้จักกันในชื่อว่า Motion Planning

Goal of Motion Planning

- Compute motion strategies, e.g.:
  - geometric paths
  - time-parameterized trajectories
  - sequence of sensor-based motion commands
- To achieve high-level goals, e.g.:
  - go to A without colliding with obstacles
  - assemble product P
  - build map of environment E
  - find object O

ตัวอย่างการวางแผนการเคลื่อนที่



A Solution

Using an RRT based motion planner program. It takes a few minute on a PC (2003).

Motion planning is not trivial:

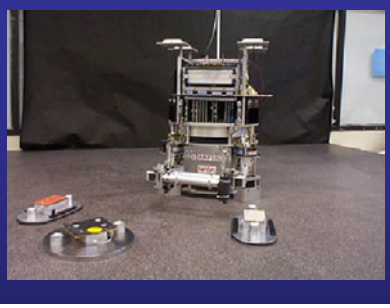
Basic Problem

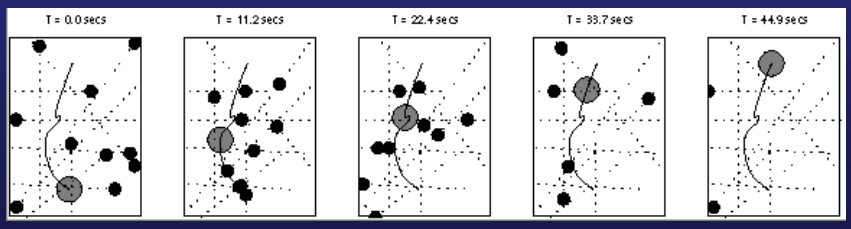
- Statement:
  Compute a collision-free path for a rigid or articulated object (the robot) among static obstacles
- Inputs:
  - Geometry of robot and obstacles
  - Kinematics of robot (degrees of freedom)
  - Initial and goal robot configurations (placements)
- Outputs:
  - Continuous sequence of collision-free robot configurations connecting the initial and goal configurations

Some Extensions to the Basic Problem

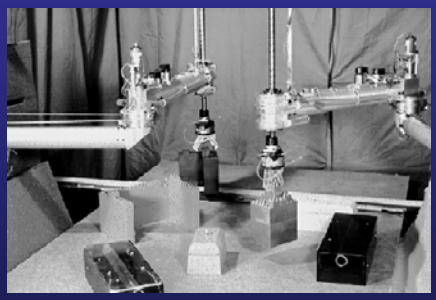
- Moving obstacles
- Multiple robots
- Movable objects
- Assembly planning
- Goal is to acquire information by sensing
  - Model building
  - Object finding/tracking
- Nonholonomic constraints
- Dynamic constraints
- Optimal planning
- Uncertainty in control and sensing
- Exploiting task mechanics (sensorless motions)
- Physical models and deformable objects
- Integration of planning and control

Moving Obstacles and Dynamic Constraints

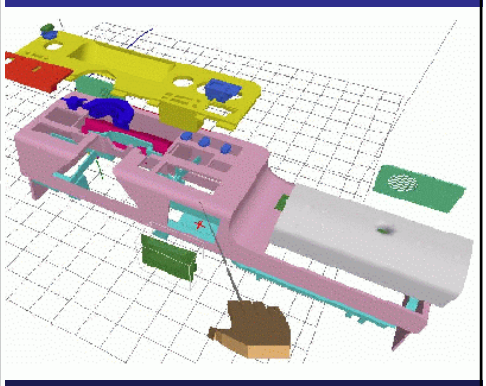




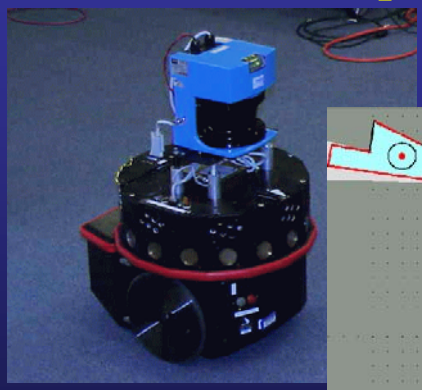
Planning in Dynamic Unpredictable Environment



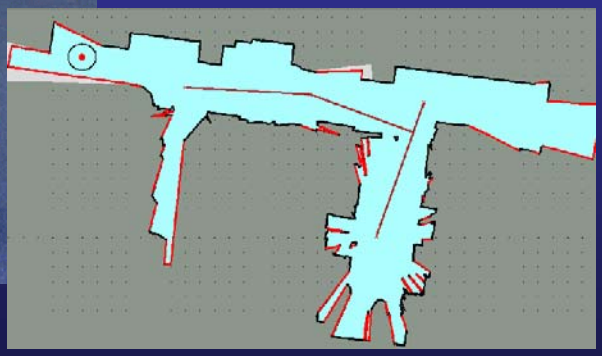
Assembly Planning



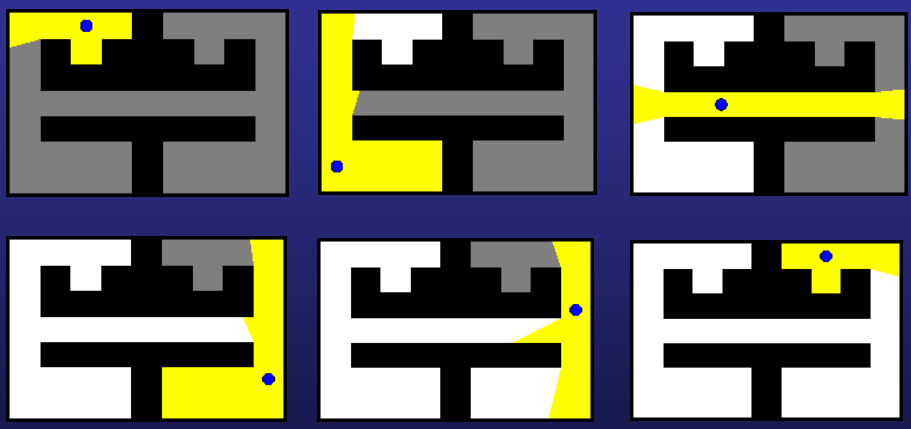
Map Building



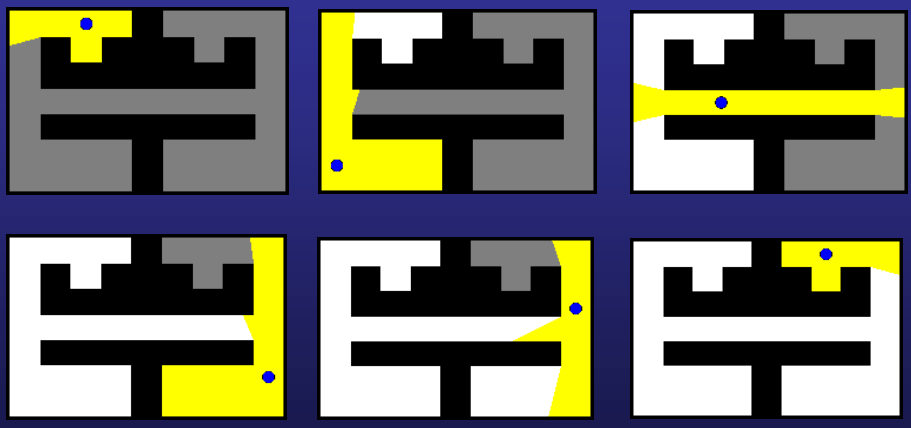
Where to move next?



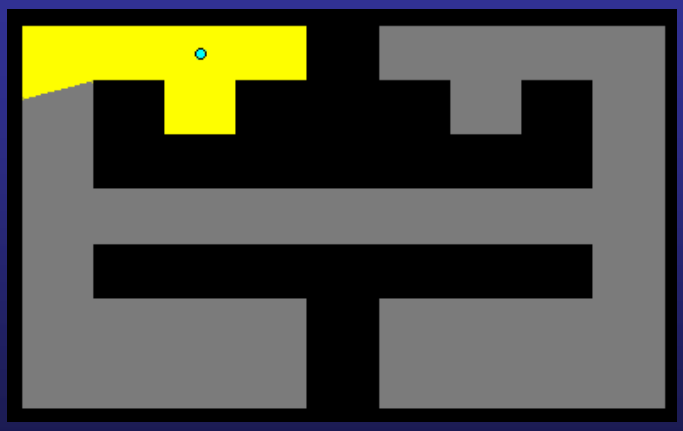
Planning Target-Finding Strategies

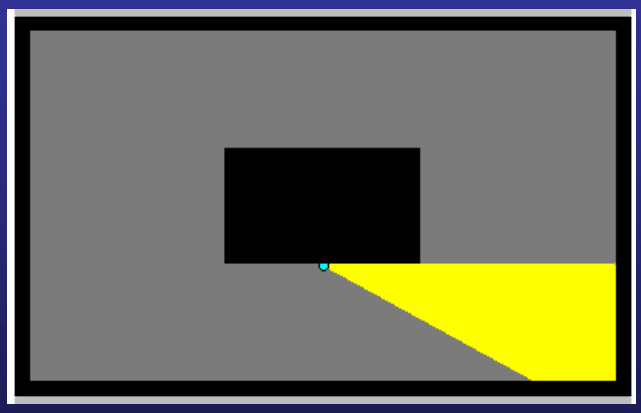


Planning Target-Finding Strategies

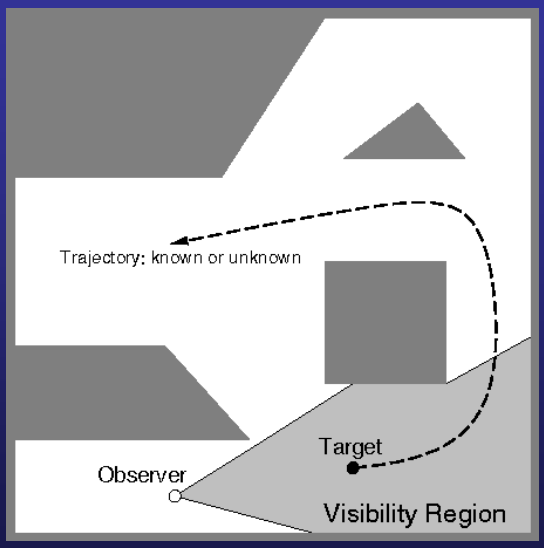


The animation shows how it works

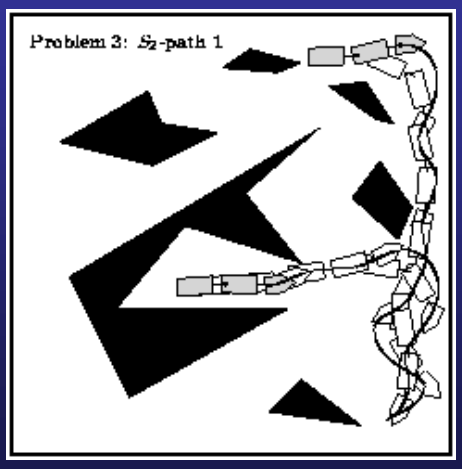




Planning Target-Tracking Strategies

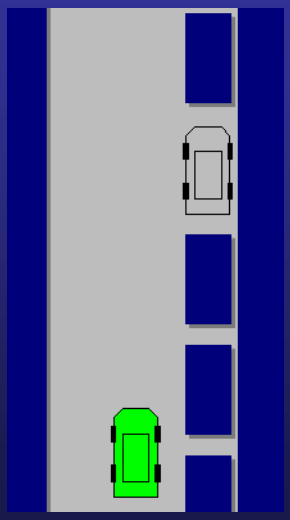


Planning for Nonholonomic Robots

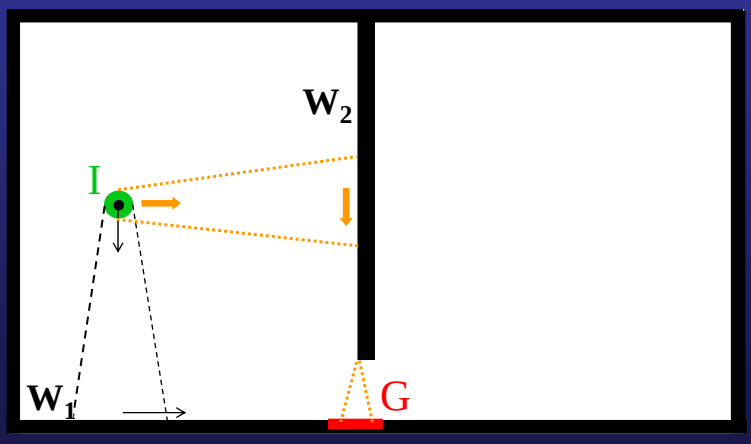


An example of nonholonomic motion planning:

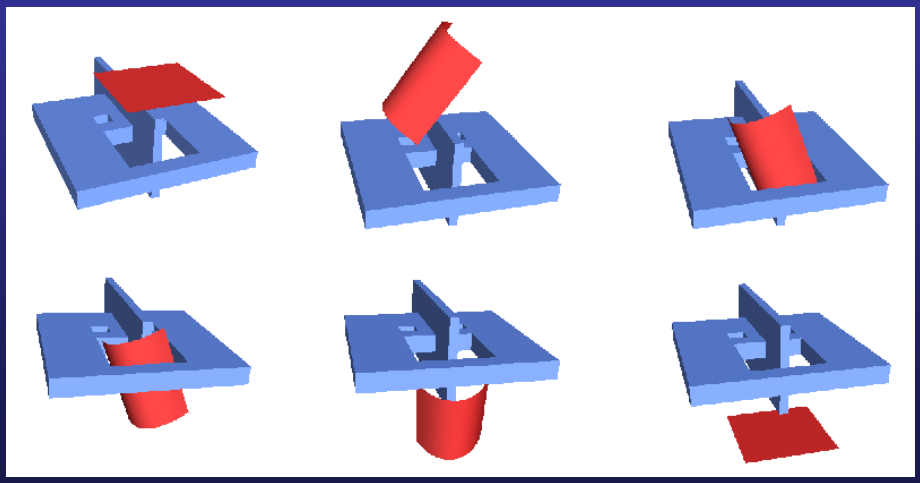
Parallel Park



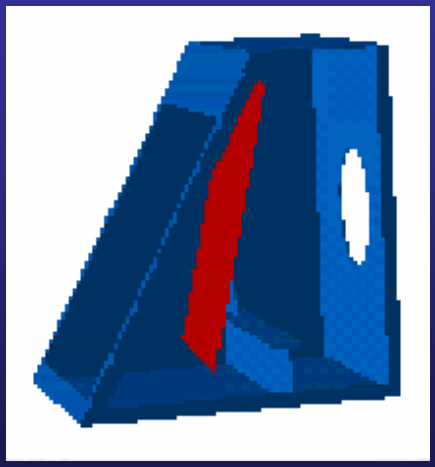
Planning with Uncertainty in Sensing and Control



Motion Planning for Deformable Objects



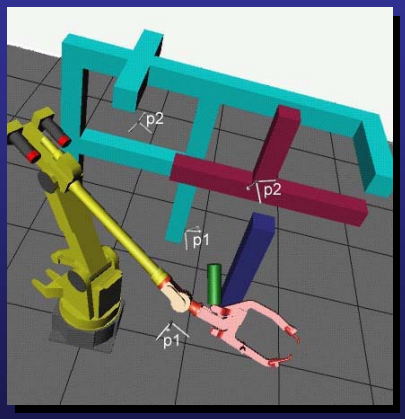
An animation shows how the flexible plate moves.

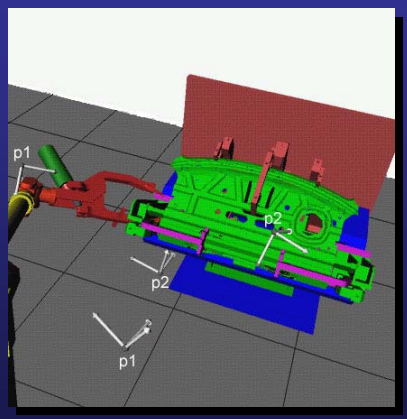


Examples of Applications

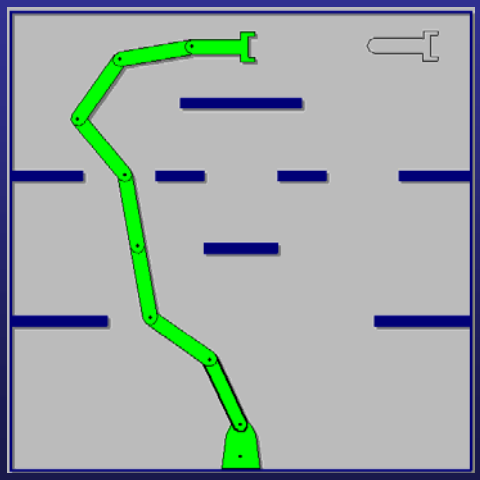
- Manufacturing:
  - Robot programming
  - Robot placement
  - Design of part feeders
- Design for manufacturing and servicing
- Design of pipe layouts and cable harnesses
- Autonomous mobile robots planetary exploration, surveillance, military scouting
- Graphic animation of “digital actors” for video games, movies, and webpages
- Medical surgery planning
- Generation of plausible molecule motions, e.g., docking and folding motions
- Building code verification

Robot Programming and Robot Placement



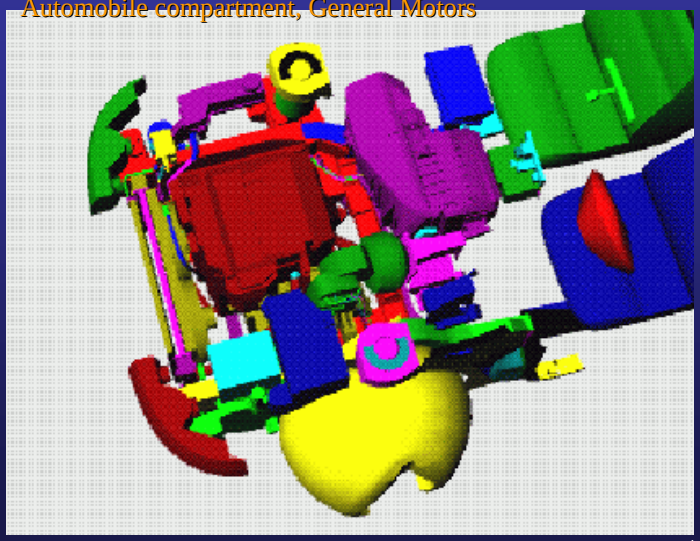


Multiple joint linkage



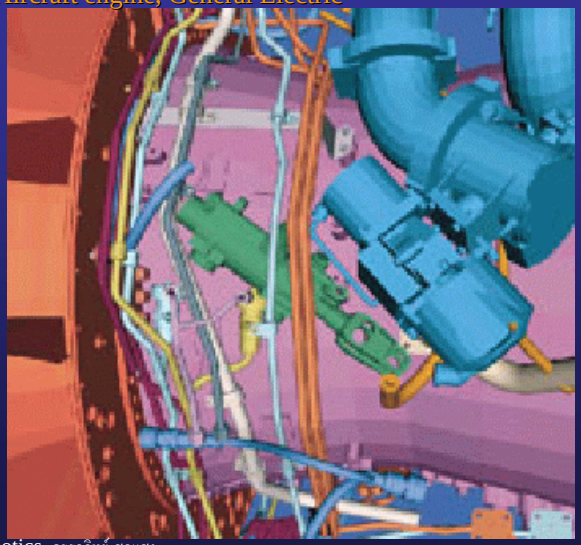
Design for Manufacturing

Automobile compartment, General Motors



Design for Maintainability

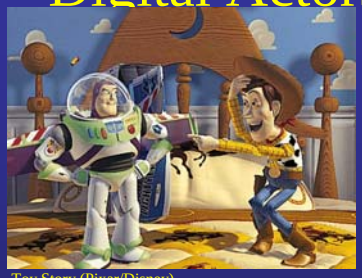
Aircraft engine, General Electric



Military Scouting in Outdoor Environment

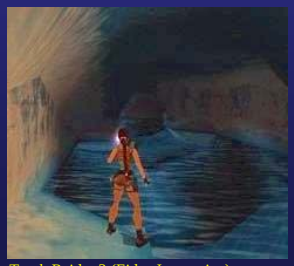
Digital Actors

A Bug's Life (Pixar/Disney)



Toy Story (Pixar/Disney)

Antz (Dreamworks)



Tomb Raider 3 (Eidos Interactive)

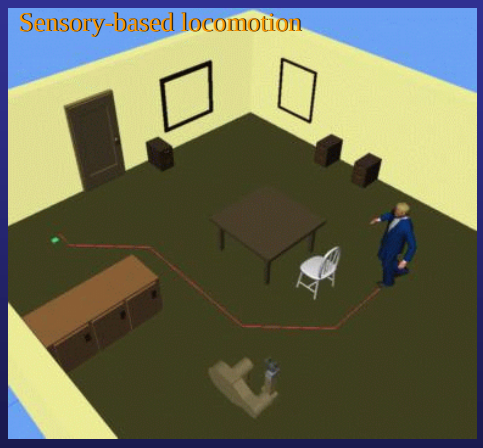
The Legend of Zelda (Nintendo)

Final Fantasy VIII (SquareOne)

Motion Planning for Digital Actors

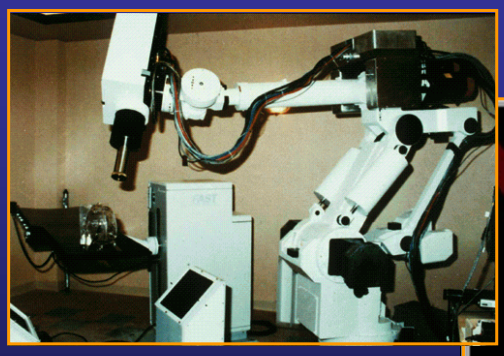
Manipulation

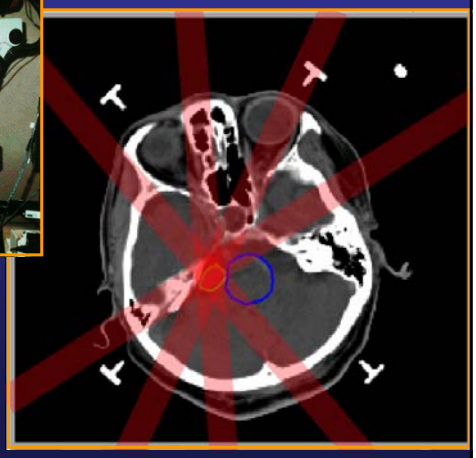
Sensory-based locomotion



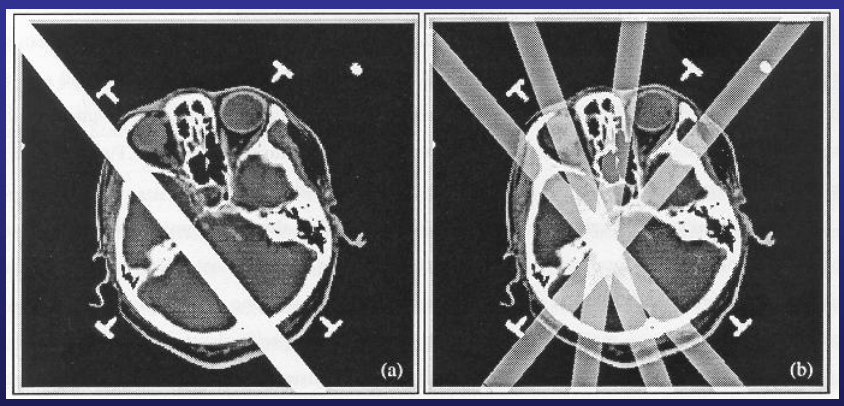
End Game 1997

Radiosurgical Planning

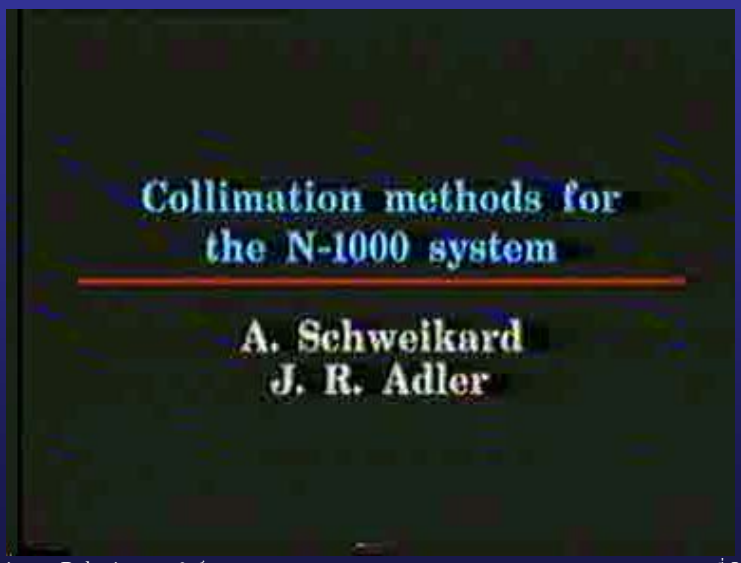




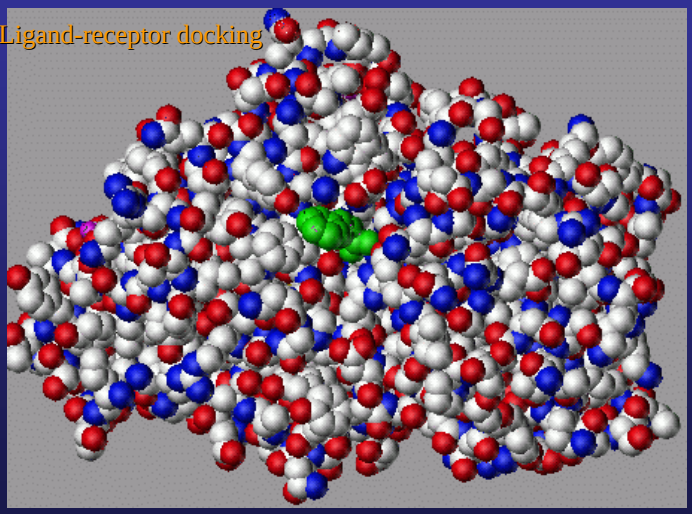
Cross-firing at a tumor while sparing healthy critical tissue



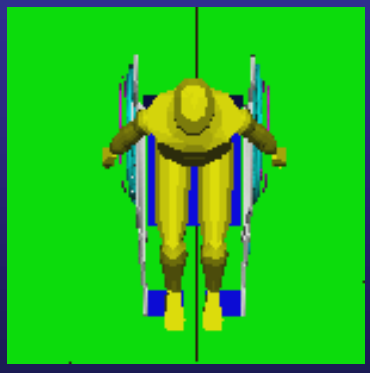
Radiosurgery



Generation of Plausible Docking Motions



Building Code Verification



Geometry review

- Analytical geometry
- Affine geometry
- Euclidean geometry

Affine Geometry

Components

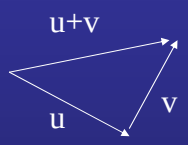
- Scalar (S)
- Point (P)
- Free vector (V)

Allowed operations

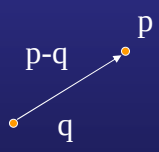
- S * V → V
- V + V → V
- P – P → V
- P + V → P

Examples

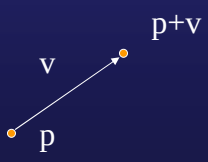
Vector addition

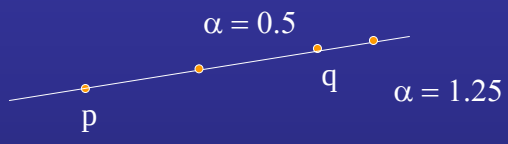


Point subtraction



Point-vector addition





จุดใดๆบนเส้นตรงที่ลากผ่านจุด p และ q สามารถสร้างได้จาก affine combination:

$$\mathbf{r} = \mathbf{p} + \alpha \cdot (\mathbf{q} - \mathbf{p})$$

Euclidean Geometry

- In affine geometry, angle and distance are not defined.
- Euclidean geometry is an extension providing an additional operation called "inner product"
- There are other types of geometry that extends affine geometry such as projective geometry, hyperbolic geometry…

Inner product is a mapping from two vectors to a real number.

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_d \end{pmatrix}, \mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_d \end{pmatrix}$$

Dot product

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^{d} u_i v_i$$

Length

$$|\mathbf{u}| = \sqrt{\mathbf{u} \cdot \mathbf{u}}$$

Distance

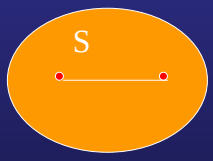
$$\text{dist}(\mathbf{P}, \mathbf{Q}) = |\mathbf{P} - \mathbf{Q}|$$

Angle

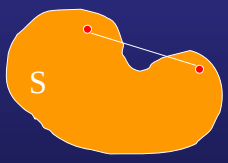
$$\text{ang}(\mathbf{u}, \mathbf{v}) = \cos^{-1}\left(\frac{\mathbf{u} \cdot \mathbf{v}}{|\mathbf{u}||\mathbf{v}|}\right)$$

Orthogonality: u and v are orthogonal when $\mathbf{u} \cdot \mathbf{v} = 0$

Let S be a set of points, S is convex when for any pair of points p and q, the line segment joining p and q is also contained in S



Convex set

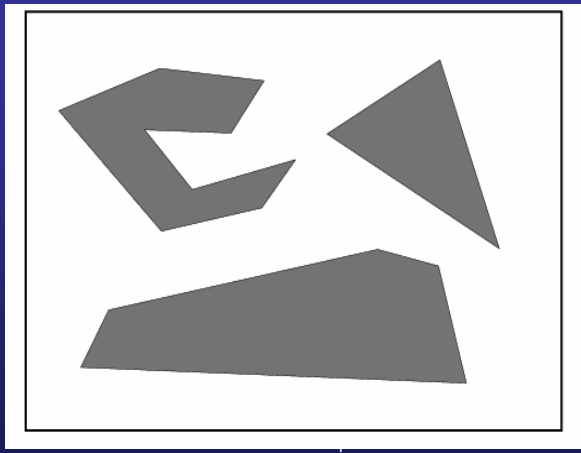


Non-convex set

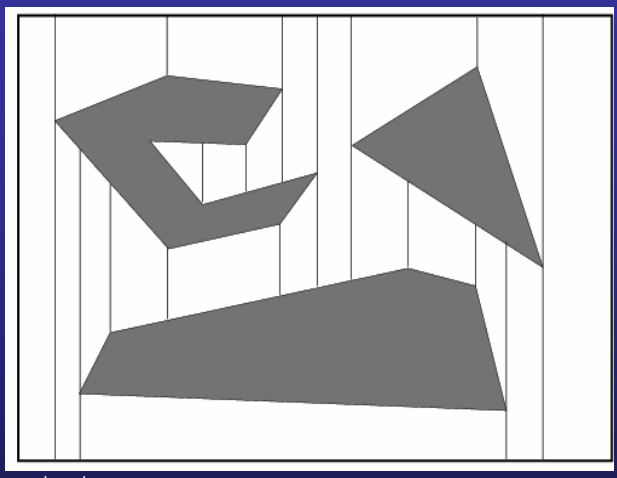
Basic Motion Planner

- Basic Concept
- Sweep Line Algorithm
- Visibility Graph
- Minkowski Sum

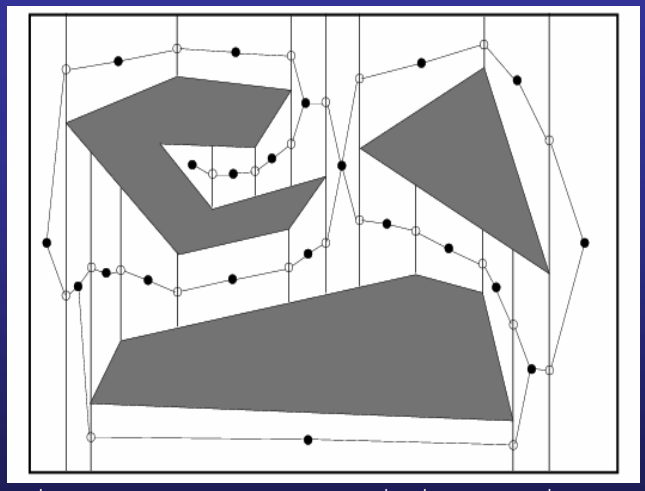
ก่อนอื่นพิจารณาการวางแผนการเคลื่อนที่หุ่นยนต์จุด



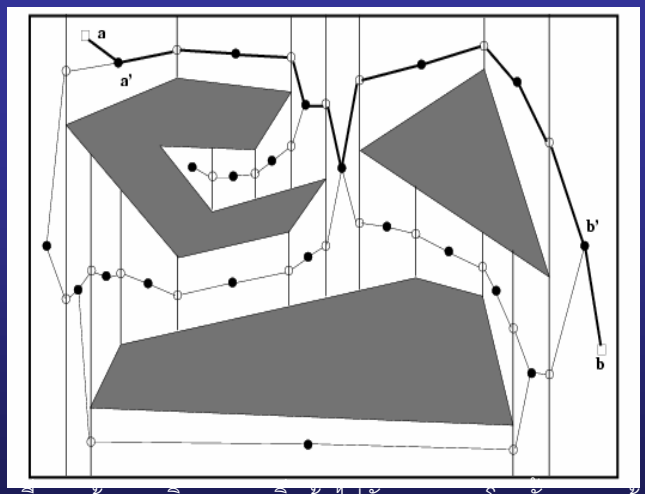
บริเวณทำงาน (workspace) แบ่งเป็นสิ่งกีดขวาง (obstacle) และบริเวณอิสระ (free space)



สร้างแผนผังสี่เหลี่ยมคางหมู (trapezoidal map) ของบริเวณอิสระด้วยการหั่นพื้นห้องที่ไม่ถูกทับด้วยสิ่งกีดขวางตามแนวตั้งออกเป็นสี่เหลี่ยมคางหมู โดยใช้พิกัดตามแกน x ของจุดยอดของสิ่งกีดขวางเป็นตำแหน่งหั่น

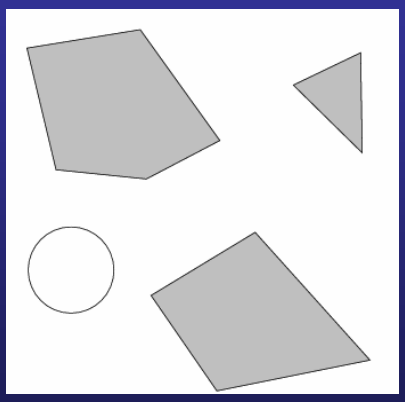


สร้างแผนที่ทางเดิน (road map) จากแผนผังสี่เหลี่ยมคางหมู ซึ่งก็คือกราฟที่บรรยายลักษณะการอยู่ติดกันของสี่เหลี่ยมคางหมูที่อยู่ในแผนผัง

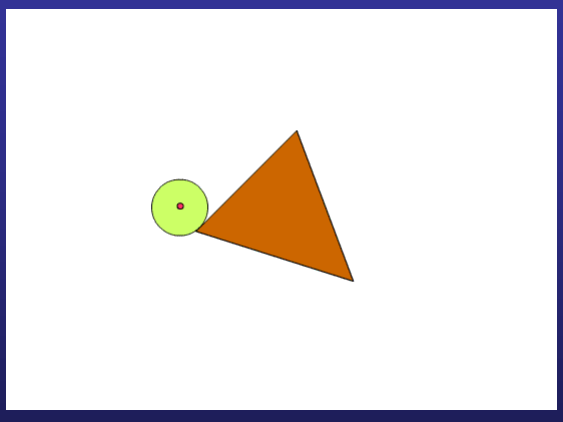


ใช้การค้น เพื่อหาเส้นทางเดินจากจุดเริ่มต้นไปยังจุดหมาย โดยขั้นแรกจะต้องหาว่าจุดเริ่มต้นและจุดหมายอยู่ในสี่เหลี่ยมคางหมูใดเมื่อทราบแล้ว จึงทำการค้นหาเส้นทางระหว่างจุดยอดของสี่เหลี่ยมคางหมูทั้งสองในแผนที่ทางเดินที่ได้จากขั้นที่สอง

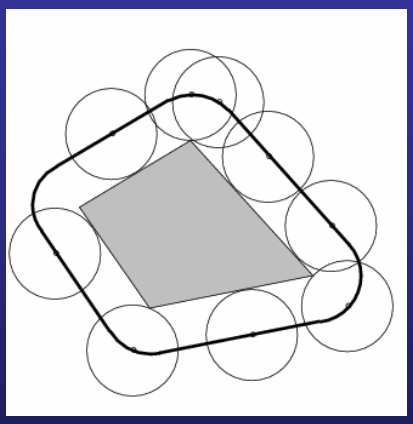
ต้องทำอย่างไรหากไม่ใช่หุ่นยนต์จุด



พิจารณาหุ่นยนต์รูปกลมในบริเวณทำงานที่มีสิ่งกีดขวางรูปหลายเหลี่ยม



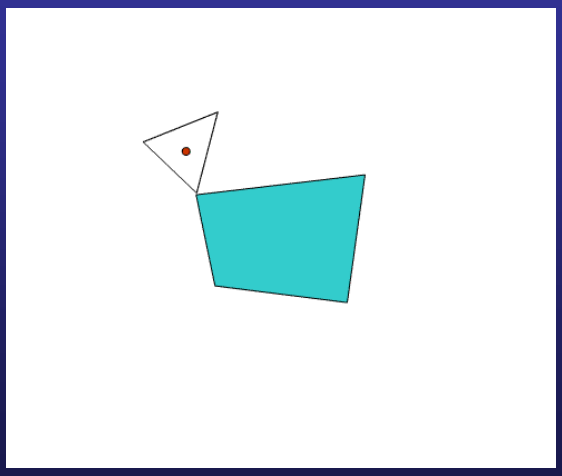
พิจารณา animation ของหุ่นยนต์รูปกลมที่เคลื่อนที่โดยไม่ชนสิ่งกีดขวาง

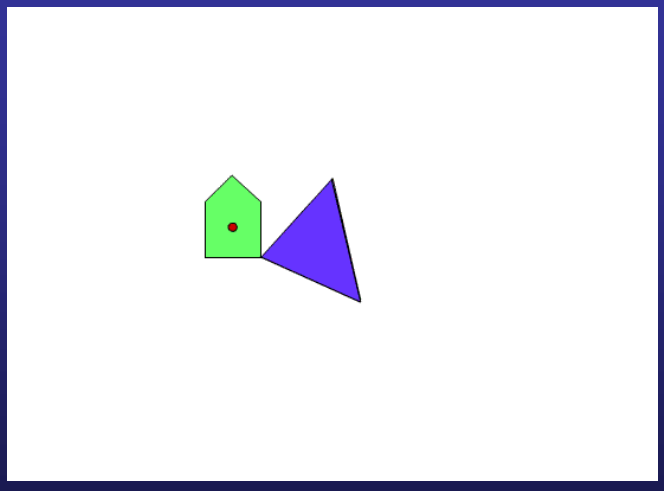


หุ่นยนต์กลมไม่ชนหรือทะลุสิ่งกีดขวางถ้าจุดศูนย์กลางของหุ่นไม่เลยเข้าไปในขอบโค้ง

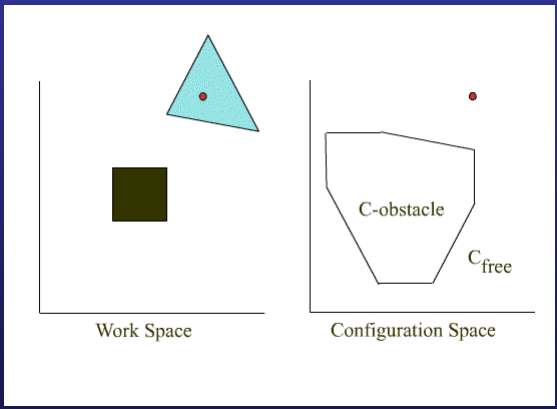
วางแผนการเคลื่อนที่ของหุ่นยนต์รูปกลมทำอย่างไรดี?

แล้วถ้าหุ่นยนต์เป็นรูปหลายเหลี่ยมที่เลื่อนได้แต่หมุนไม่ได้ล่ะ?

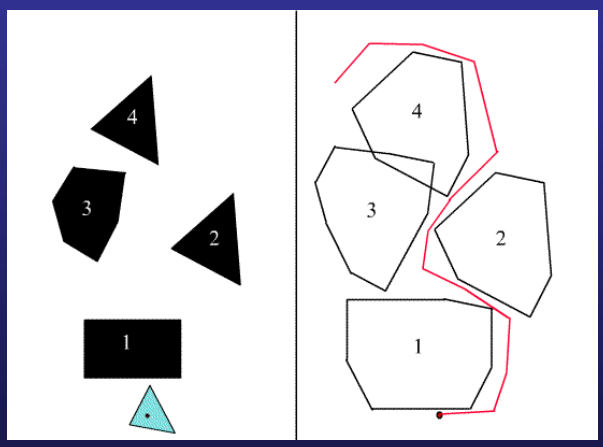




C-Obstacle



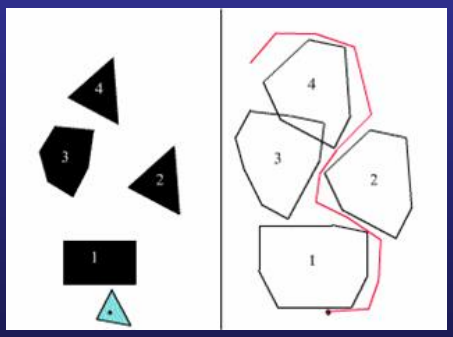
แล้วสิ่งกีดขวางหลายชิ้นล่ะ



สรุป

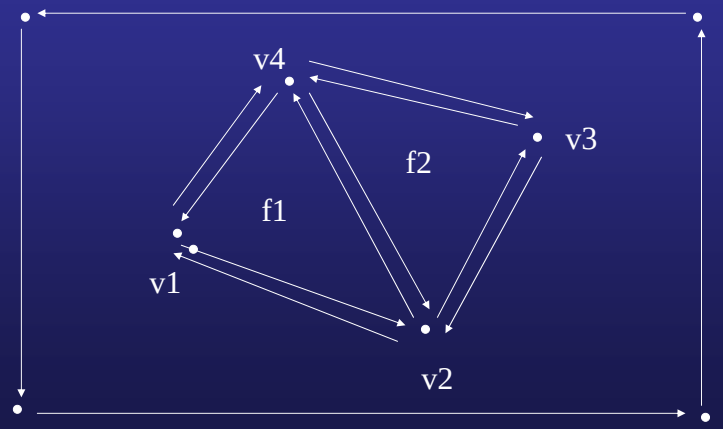
- สำหรับสิ่งกีดขวางและหุ่นยนต์รูปหลายเหลี่ยมคอนเวกซ์ เราสามารถสร้างสิ่งกีดขวางใหม่ในอีกระนาบ (ที่เราจะเรียกว่า C-Space) ทำให้เราวางแผนการเคลื่อนที่ได้โดยมองหุ่นยนต์เป็นจุดและใช้การวางแผนการเคลื่อนที่ของหุ่นยนต์จุดที่กล่าวไปแล้ว
- สิ่งกีดขวางใหม่ในระนาบ C-Space สามารถคำนวณได้โดยใช้ผลรวมมิงคอฟสกี (Minkowski sum)

Implementation



ออกแบบ data structure ที่ใช้แทน polygon แล้วทำให้เราสามารถทำ operation ที่ต้องการเช่น union, intersection ได้อย่างมีประสิทธิภาพ

Doubly-connected edge list



Components

Vertex table

Vertex	Coordinates	Incident Edge

Face table

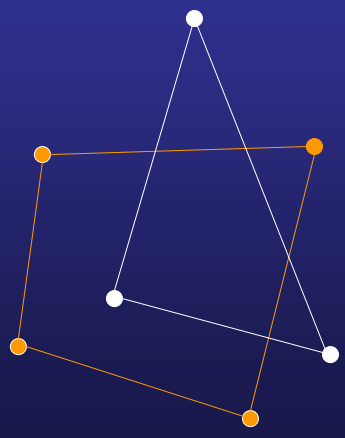
Face	Outer	Inner

Components (cont)

Half-edge table

Half-edge	Origin	Twin	Incident face	Next	Prev

Intersection and union of polygons

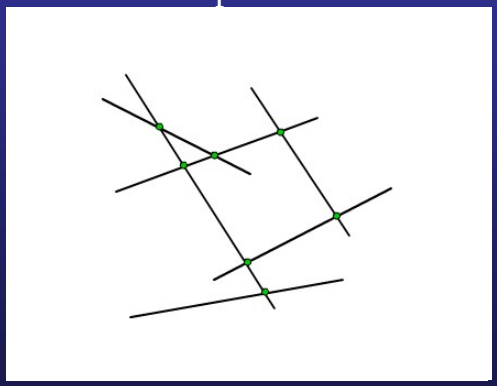


Sweep line algorithm

- Sweep (or rotate) a line through all inputs.
- Partially construct the result during the sweep.
- Previously constructed result is not affected by the un-swept input.
- Using event points to mark when to do what.

Line segment intersection

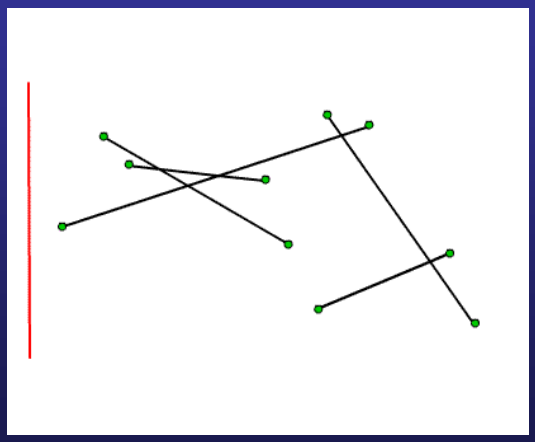
Input: n line segments
Output: all intersection points



A naïve algorithm takes $O(n^2)$ to compute intersection points for all pairs of n segments.

We can do better…

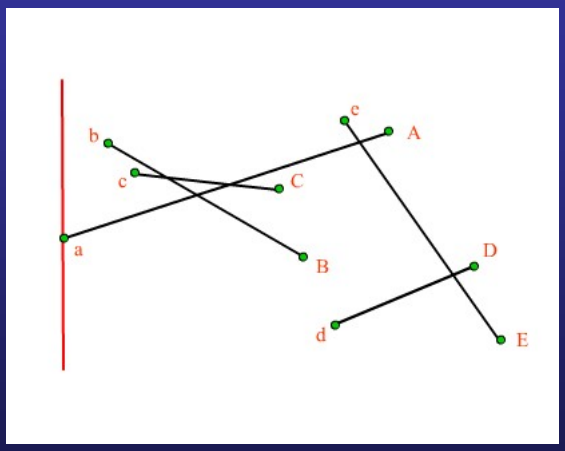
Sweeping…



Let's trace…

Intersect:

aA

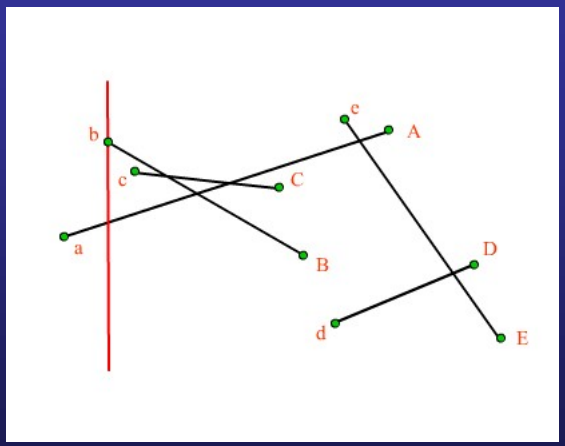


Event: a b c C B d e A D E

Let's trace…

Intersect:

aA



Insert ab
Add bB

Event: b c C B d e A D E

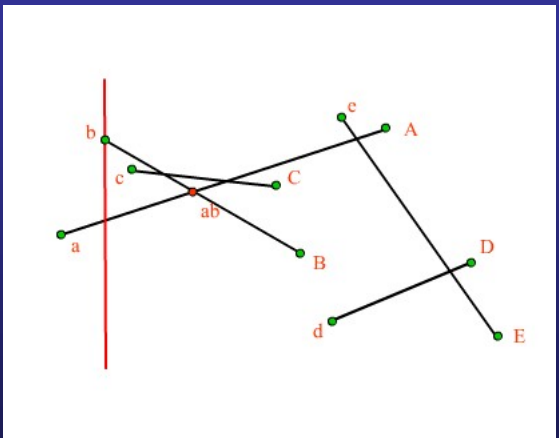
Key: two segments intersect, they must be adjacent in the intersection list at certain moment.

Let's trace…

Intersect:

bB

aA



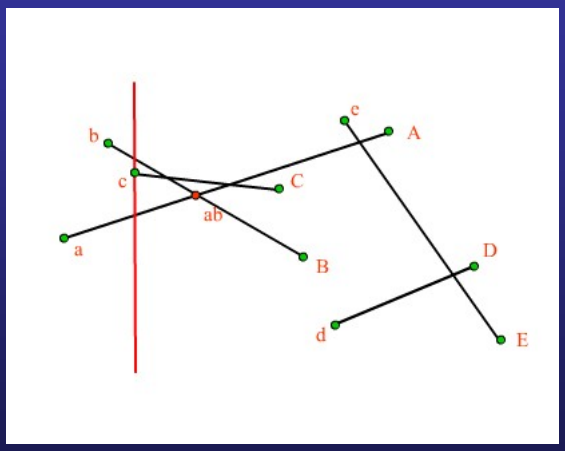
Event: b c ab C B d e A D E

Let's trace …

Intersect:

bB

aA



Insert bc
Insert ac
Add cC

Event: c ab C B d e A D E

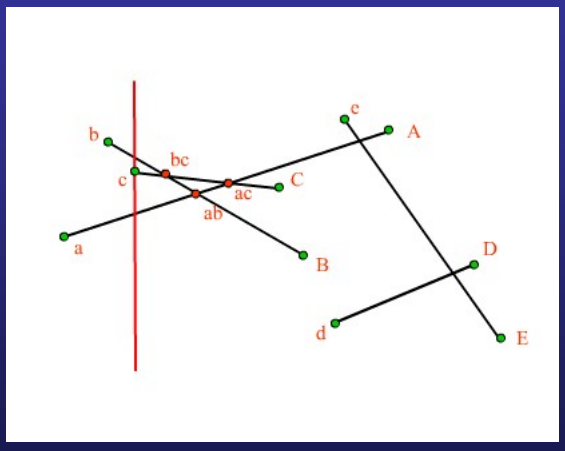
Let's trace …

Intersect:

bB

cC

aA



Event: c bc ab ac C B d e A D E

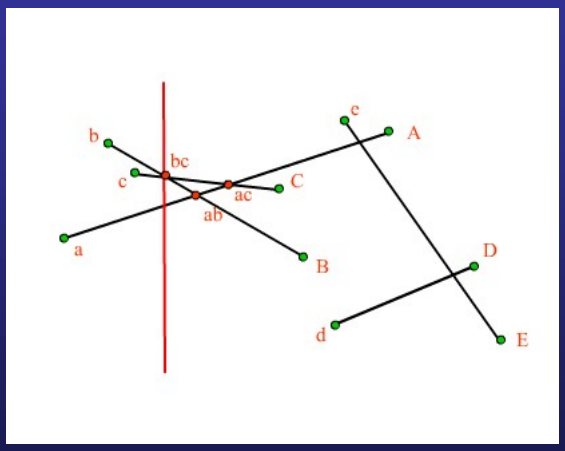
Let's trace …

Intersect:

bB

cC

aA



Count bc

Swap bB-cC

Event: bc ab ac C B d e A D E

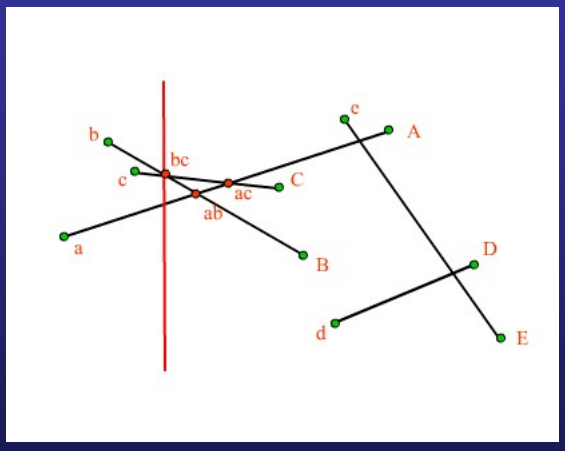
Let's trace …

Intersect:

cC

bB

aA



Event: bc ab ac C B d e A D E

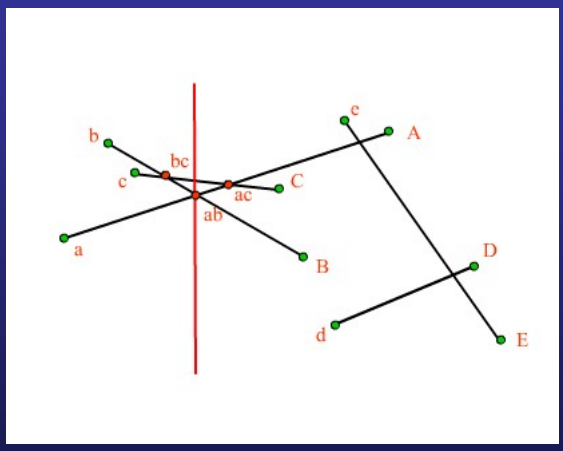
Let's trace …

Intersect:

cC

bB

aA



Count ab
Swap aA-bB

Event: ab ac C B d e A D E

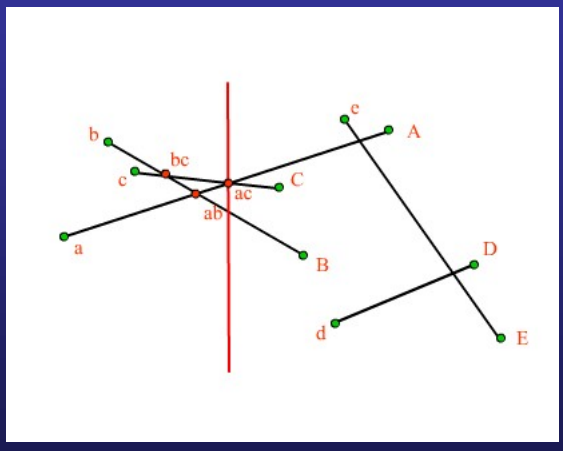
Let's trace …

Intersect:

cC

aA

bB



Count ac
Swap aA-cC

Event: ac C B d e A D E

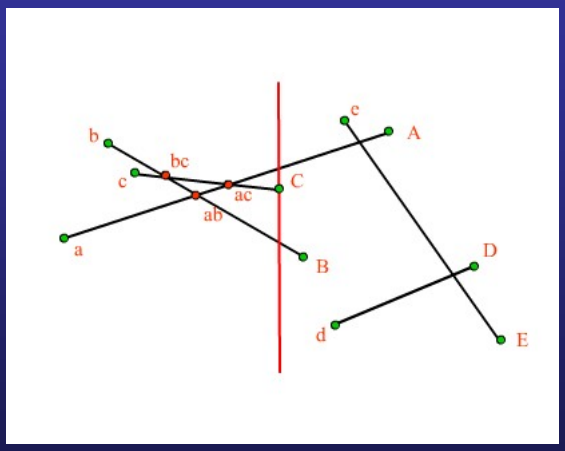
Let's trace …

Intersect:

aA

cC

bB



Remove cC

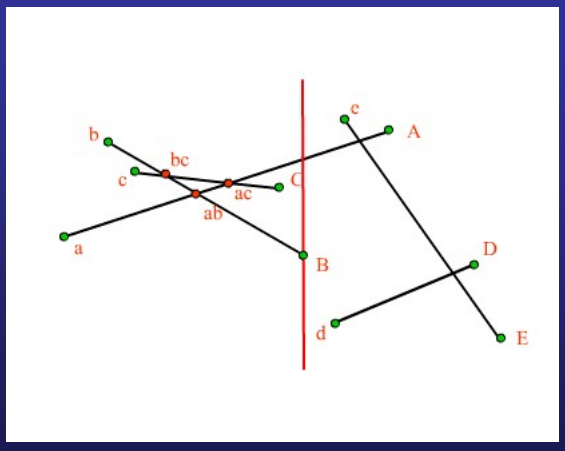
Event: C B d e A D E

Let's trace …

Intersect:

aA

bB



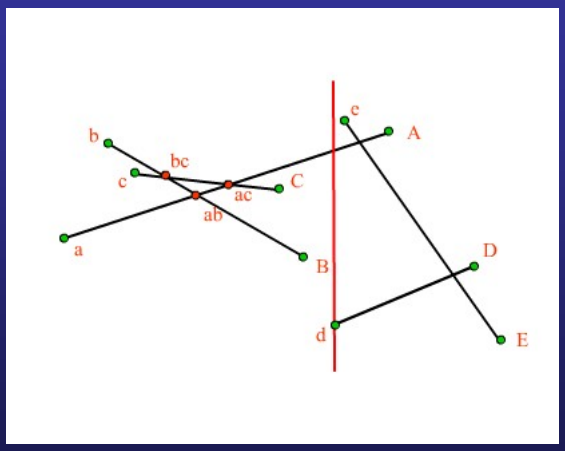
Remove bB

Event: B d e A D E

Let's trace …

Intersect:

aA



Add dD

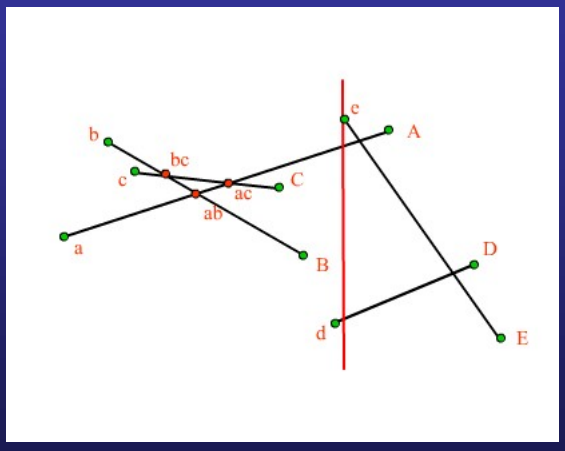
Event: d e A D E

Let's trace …

Intersect:

aA

dD



Add eE
Insert ae

Event: e A D E

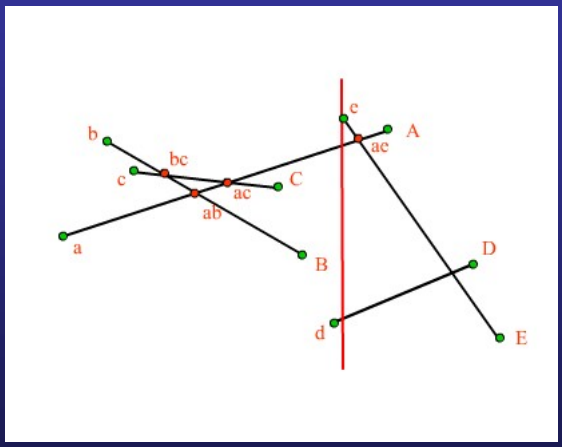
Let’s trace …

Intersect:

eE

aA

dD



Event: e ae A D E

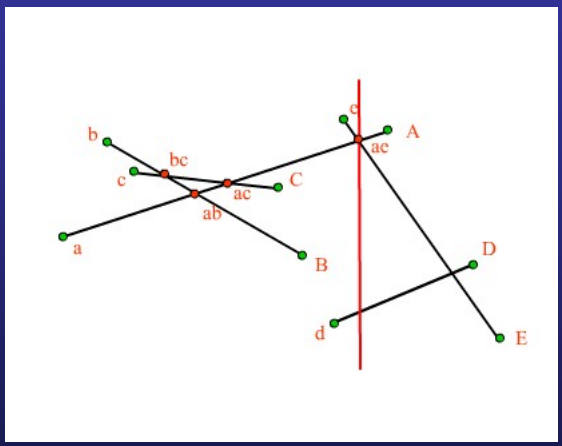
Let’s trace …

Intersect:

eE

aA

dD



Count ae
Swap eE-aA
Insert de

Event: ae A D E

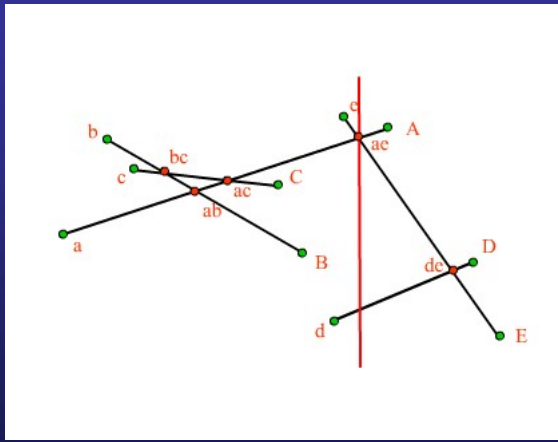
Let's trace …

Intersect:

aA

eE

dD



Event: ae A de D E

Let's trace …

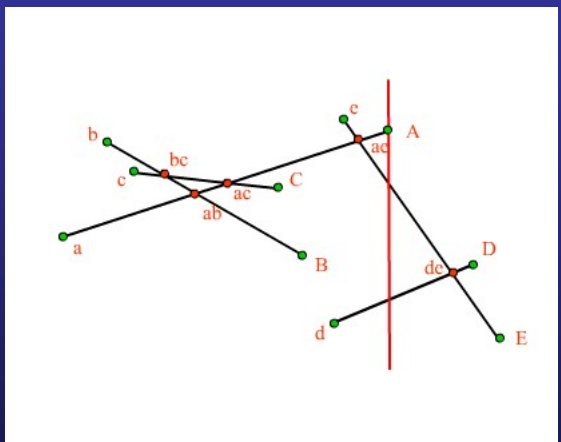
Intersect:

aA

eE

dD

Remove aA



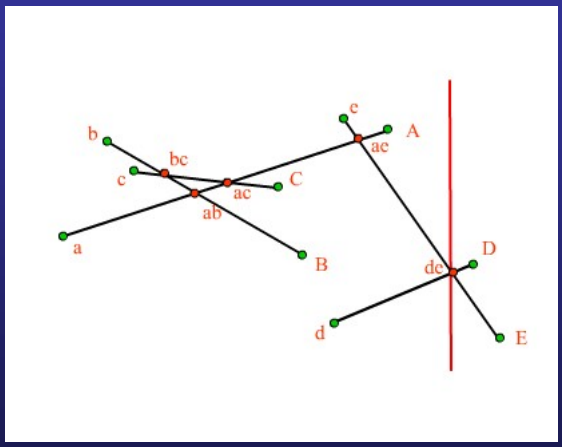
Event: A de D E

Let's trace …

Intersect:

eE

dD



Count de
Swap dD-eE

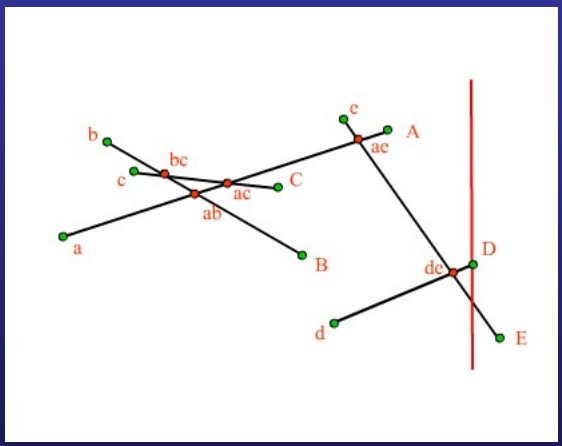
Event: de D E

Let's trace …

Intersect:

dD

eE



Remove dD

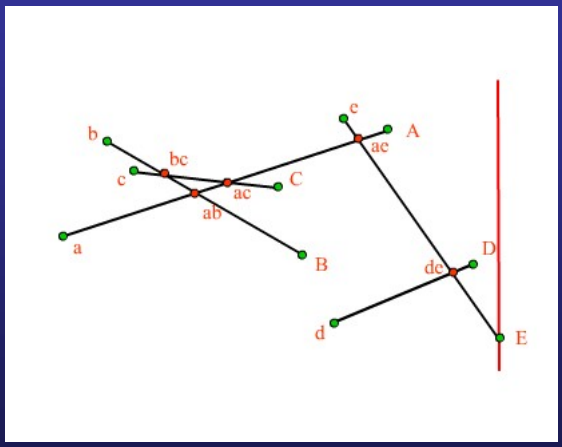
Event: D E

Let's trace …

Intersect:

eE

Remove eE

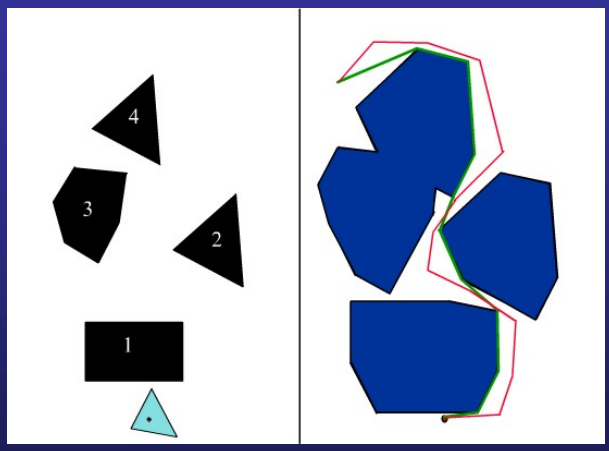


Event: E

Analysis

- Sorting points takes O(n lg(n)).
- Event queue is implemented using heap: insertion, deletion take O(lg(n)).
- Intersection list is implemented using balanced binary tree: insertion, deletion take O(lg(n))
- Swapping can occur only at intersection event points, suppose there are I intersection events.
- The algorithm therefore runs in O(nlg(n) + I lg(n)).

Visibility Graph

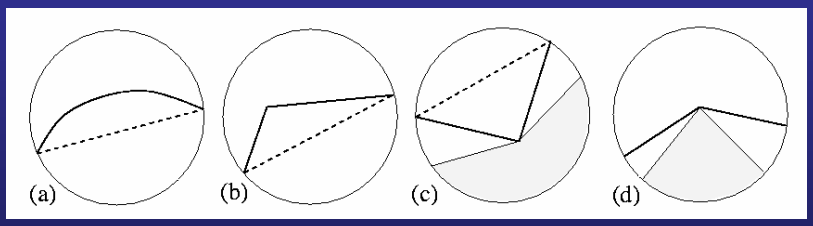


Is this the shortest path possible?

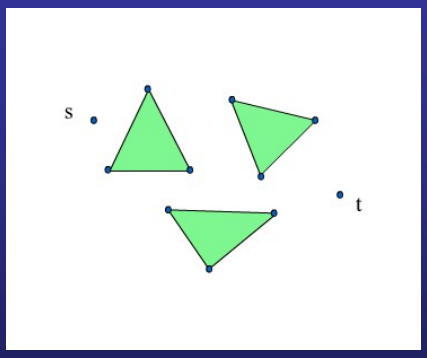
The shortest path between any two points s and t that avoids a set of polygonal obstacles is a polygonal curve, whose vertices are either vertices of the obstacles or the points s and t.

Proof

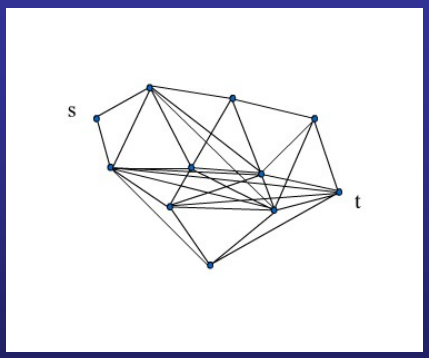
Zooming in…



Definition: The visibility graph of s and t and the obstacle set is a graph whose vertices are s, t and the obstacle vertices, and vertices v and w are joined by an edge if v and w are either mutually visible or if (v,w) is an edge of some obstacle.

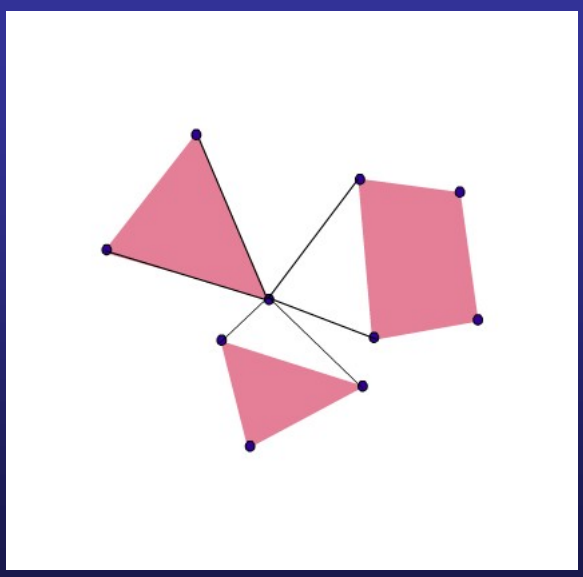


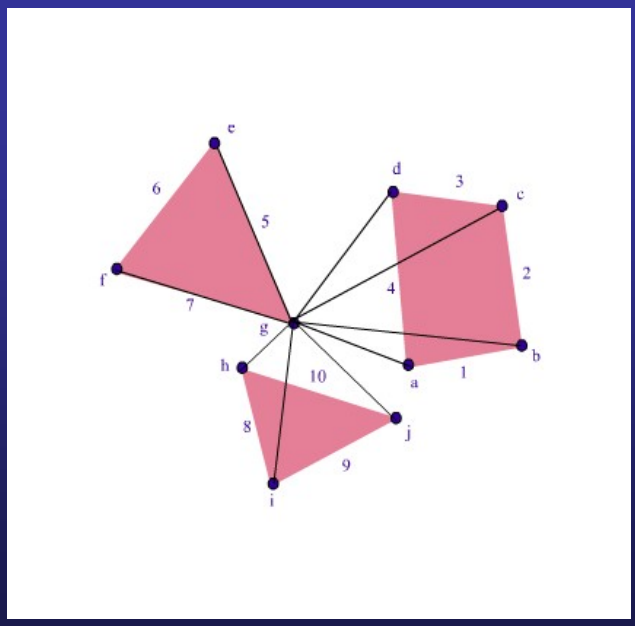
Obstacles with vertices s and t

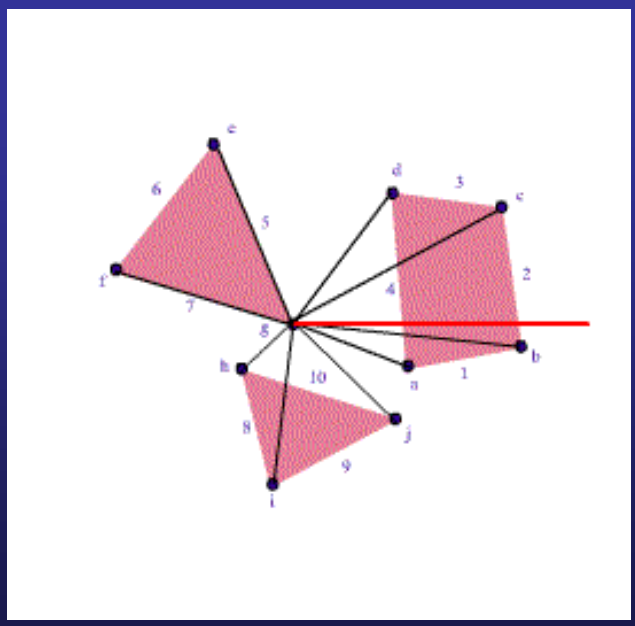


Corresponding visibility graph

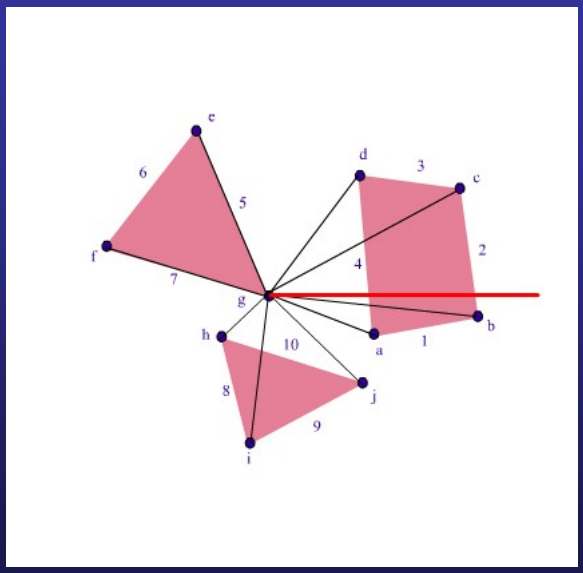
Computing Visibility Graph




c
d
3
c
6
5
f
4
2
7
g
b
h
10
a
1
j
8
9
i

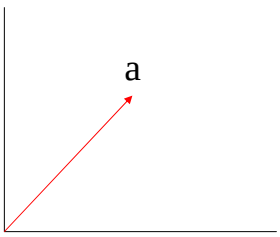

e
d
3
c
6
5
f
4
2
7
g
b
h
10
a
1
j
8
9
i

Let's trace…

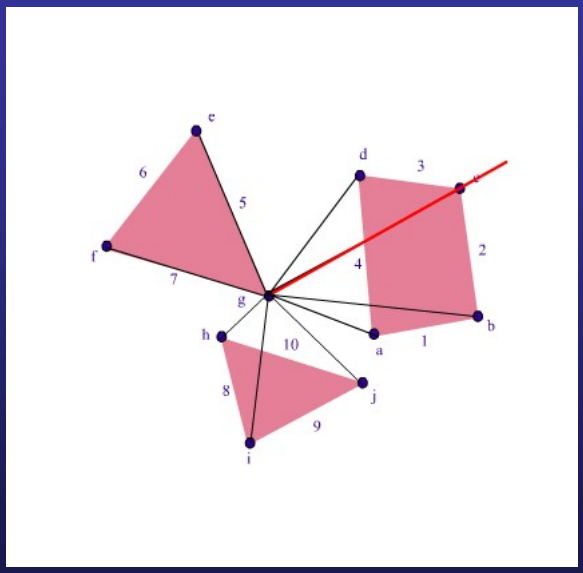


Sorting event points by their angles takes O(nlg(n)).

Computing angle of vector a with respect to the x-axis…

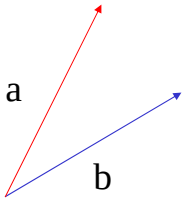


$$\mathbf{a} = |\mathbf{a}|(\cos\theta, \sin\theta)$$

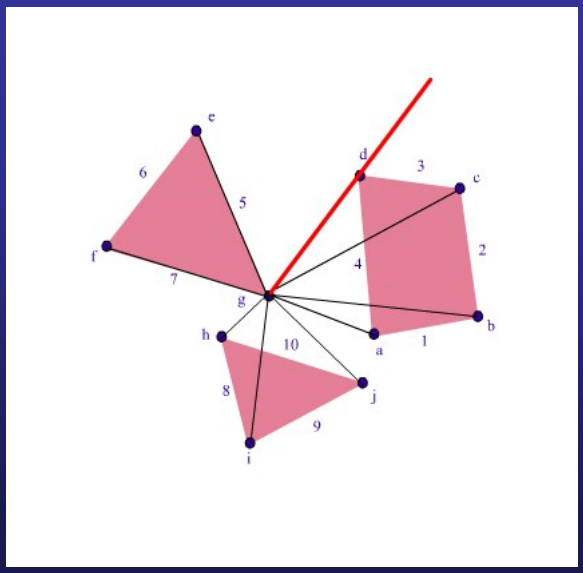


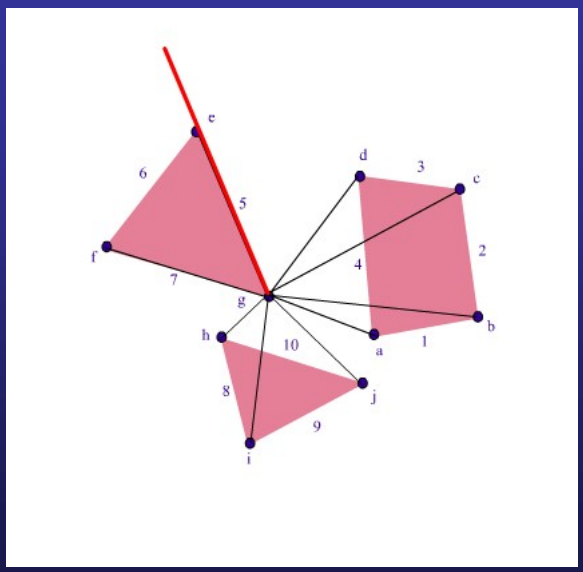
Orientation Checking…

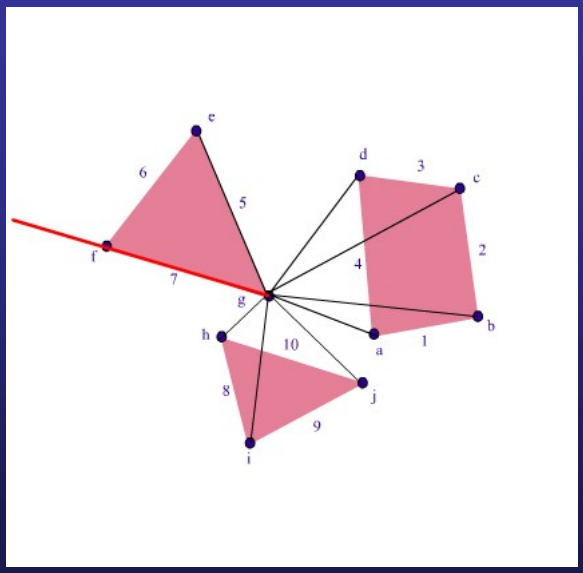
Vector a is in the counterclockwise side of vector b iff |b x a| > 0.

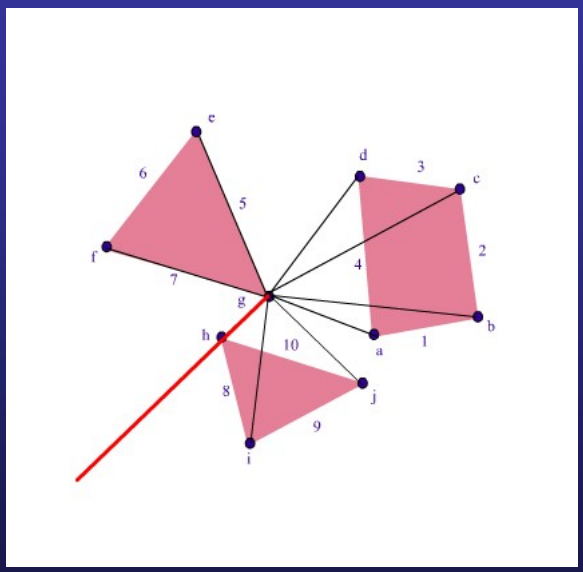


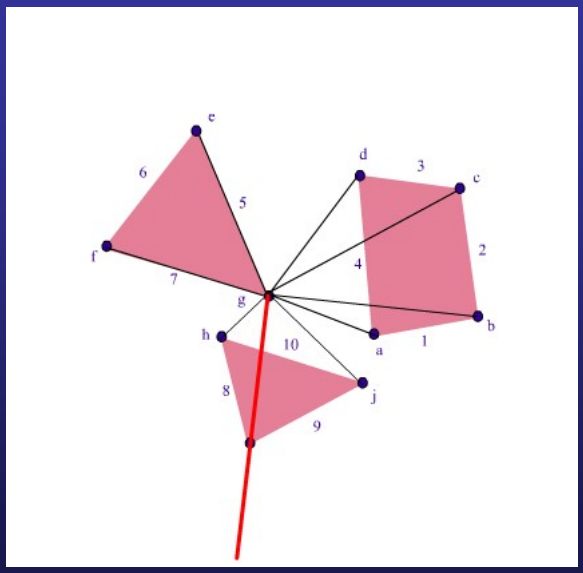
$$\mathbf{a}\times\mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_x & a_y & 0 \\ b_x & b_y & 0 \end{vmatrix} = (a_x b_y - a_y b_x)\mathbf{k}$$

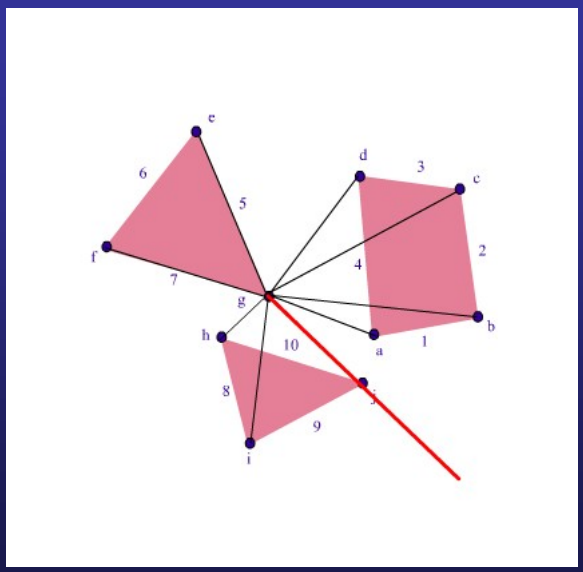

e
d
c
6
5
3
2
f
4
7
g
b
h
10
a
1
8
j
9
i

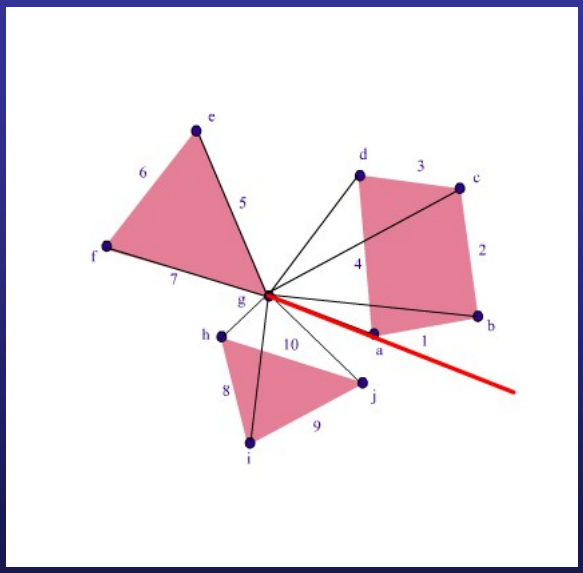

e
d
c
6
5
3
2
f
4
7
g
b
h
10
a
1
8
j
9
i

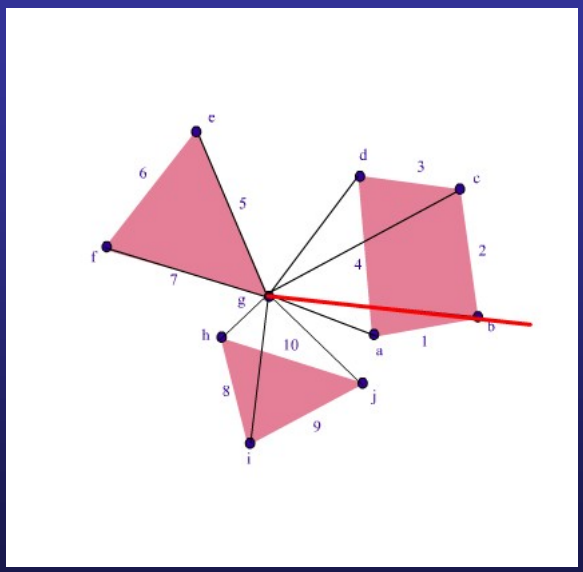

e
d
c
6
3
5
2
f
4
7
g
b
h
a
1
10
8
j
9
i


e
d
c
6
3
5
2
f
4
7
g
b
h
a
1
10
8
j
9
i


e
d
c
6
5
3
2
f
4
7
g
b
h
10
a
1
8
j
9


e
d
c
6
5
3
2
f
4
7
g
b
h
10
a
1
8
9
i

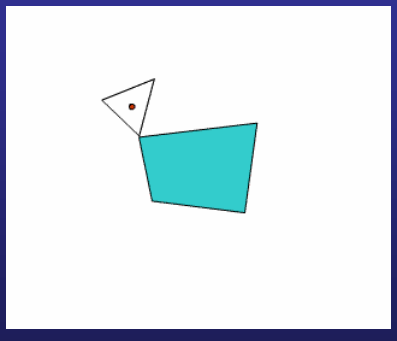

e
d
c
6
5
3
2
f
4
7
g
b
h
10
a
1
8
j
9
i

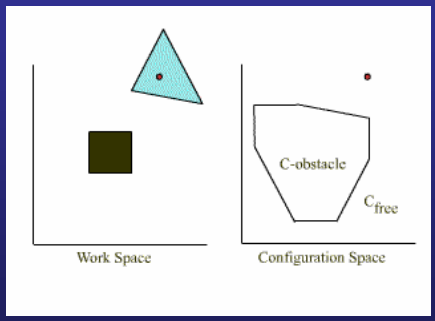

e
d
c
6
5
3
2
f
4
7
g
b
h
10
a
1
8
j
9
i

Analysis

- Sorting takes O(n lg(n))
- Computing intersection at θ=0 takes O(n)
- At each event, insertion, deletion take O(lg(n)) using balanced binary tree
- We have to repeat all these steps for all n points, therefore the entire computation takes $O(n^2 lg(n))$.

Minkowski Sum





เราสามารถคำนวณสิ่งกีดขวางใน C-Space ได้โดยใช้ Minkowski Sum

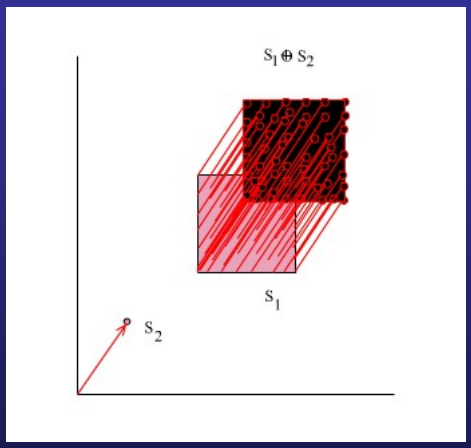
นิยาม

สำหรับเซตของเวกเตอร์ A และ B ใดๆ นั้น ผลรวมมิงคอฟสกี ของA และ B เขียนแทนได้ด้วย $A \oplus B$ และมีนิยามคือ

$$A \oplus B = \{\mathbf{a} + \mathbf{b} \mid \mathbf{a} \in A \ \wedge \ \mathbf{b} \in B\}$$

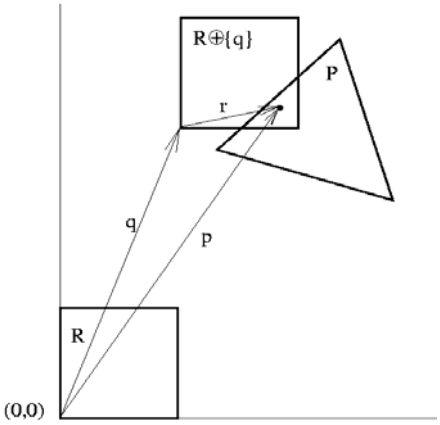
ซึ่งก็คือเซตของเวกเตอร์ทั้งหมดที่ได้จากการบวกคู่ของเวกเตอร์ใดๆ โดยเวกเตอร์หนึ่งมาจากเซต *A* และอีกเวกเตอร์มาจากเซต *B*

Minkowski sum of S_1 and S_2



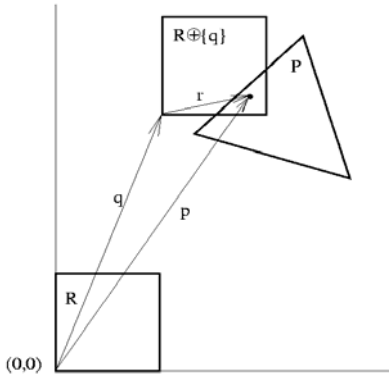
ถ้า S_2 = { **q** } เหมือนกับการเลื่อน S_1 ไปด้วยเวกเตอร์ **q**

พิจารณาหุ่นยนต์ *R* และสิ่งกีดขวาง *P*



หุ่นยนต์ *R* ที่เลื่อนไปด้วยเวกเตอร์ q เรียกว่า หุ่นยนต์ *R* ที่ configuration q

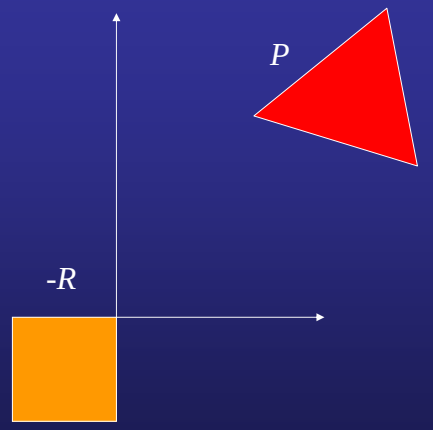
Configuration ต้องห้ามคือ configuration ที่เลื่อนหุ่นไปทับกับสิ่งกีดขวาง



เรารู้ว่า q เป็นคอนฟิกุเรชันต้องห้ามก็ต่อเมื่อมีบางเวกเตอร์ $\mathbf{r} \in R$ และ $\mathbf{p} \in P$ ที่ทำให้ q + r = p หรือเราจะกล่าวได้ว่าสำหรับ คู่เวกเตอร์ $\mathbf{r} \in R$ และ $\mathbf{p} \in P$ ใดๆ จะได้ว่า q = p – r เป็นคอนฟิกุเรชันต้องห้าม นั่นก็คือเราจะได้ว่า C-Obstacle $= P \oplus (-R)$

โดยที่ $-R = \{-\mathbf{a} \mid \mathbf{a} \in R\}$

คำนวณผลรวมมิงคอฟสกี



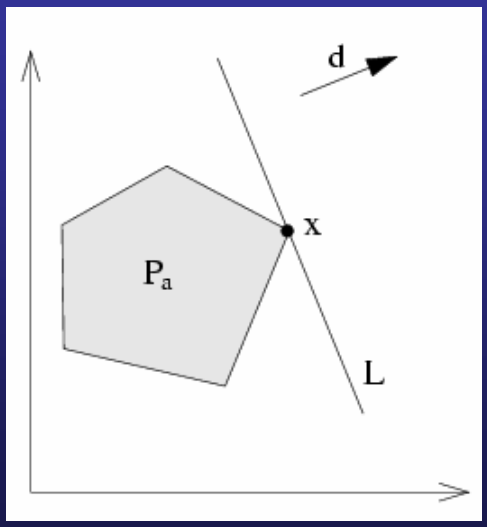
Naïve approach: บวกทุกคู่ของจุดยอดของ P กับ R แล้วหา convex hull ของผลลัพธ์ที่ได้

Lemma 1: กำหนดให้ A และ B เป็นเซตของเวกเตอร์ทั้งหมดจากจุดกำเนิดไปยังแต่ละจุดในรูปหลายเหลี่ยมคอนเวกซ์สองรูป เราจะได้ว่า $A \oplus B$ เป็นเซตของเวกเตอร์ที่ชี้ไปยังจุดทั้งหมดในรูปหลายเหลี่ยมแบบคอนเวกซ์เช่นกัน

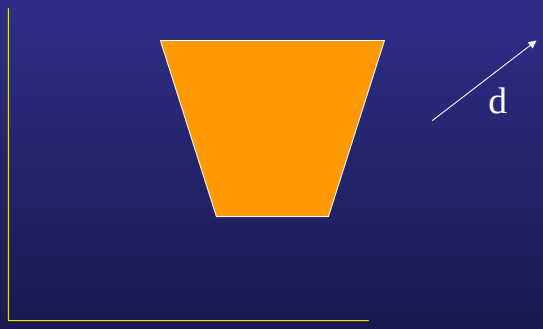
พิสูจน์ ให้ $C = A \oplus B$ และสมมุติขัดแย้งว่า C ไม่เป็นเซตคอนเวกซ์ นั่นคือมี $\mathbf{c}_1, \mathbf{c}_2 \in C$ ซึ่งเวกเตอร์ทั้งหมดที่ชี้ไปยังส่วนของเส้นตรง $\mathbf{c}_1\mathbf{c}_2$ ไม่ได้อยู่ใน C ทั้งหมด ให้ $\mathbf{c}_1 = \mathbf{a}_1 + \mathbf{b}_1$ และ $\mathbf{c}_2 = \mathbf{a}_2 + \mathbf{b}_2$ โดยที่ $\mathbf{a}_1, \mathbf{a}_2 \in A$ และ $\mathbf{b}_1, \mathbf{b}_2 \in B$ (เพราะ $\mathbf{c}_1, \mathbf{c}_2 \in C$ จึงต้องมี $\mathbf{a}_1, \mathbf{a}_2, \mathbf{b}_1, \mathbf{b}_2$ อยู่จริง)

กำหนดให้ $\mathbf{c}' = \mathbf{c}_1 + \lambda(\mathbf{c}_2 - \mathbf{c}_1)$ โดยที่ $\lambda \in [0,1]$ เป็นเวกเตอร์ที่ชี้ไปยังจุดบนส่วนของเส้นตรง $\mathbf{c}_1\mathbf{c}_2$ เราเขียนใหม่ได้ว่า $\mathbf{c}' = \mathbf{a}' + \mathbf{b}'$ โดยที่ $\mathbf{a}' = \mathbf{a}_1 + \lambda(\mathbf{a}_2 - \mathbf{a}_1)$ และ $\mathbf{b}' = \mathbf{b}_1 + \lambda(\mathbf{b}_2 - \mathbf{b}_1)$ แต่ว่า $\mathbf{a}' \in A$ และ $\mathbf{b}' \in B$ เพราะ A และ B เป็นเซตคอนเวกซ์ ดังนั้นได้ว่า $\mathbf{c}'$ อยู่ใน $A \oplus B$ □

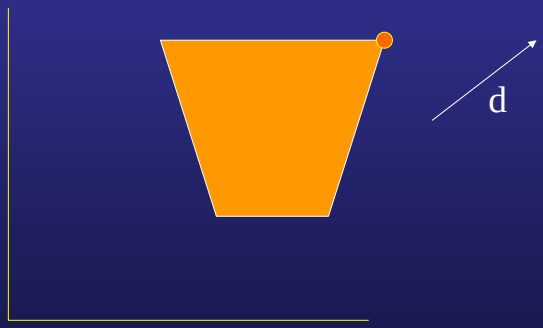
จุดสุดขีด (extreme point) ของ P_a ในทิศทาง d



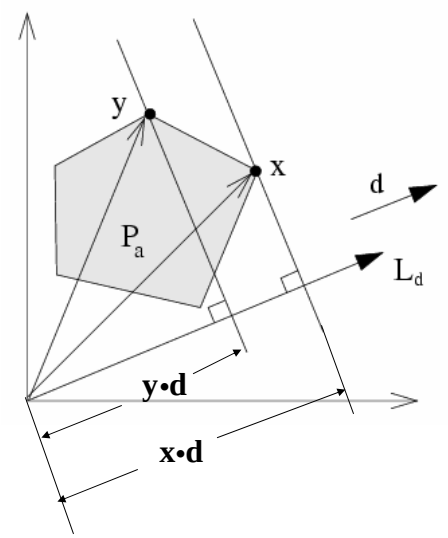
Definition: given a vector **d**, a point **p** is an **extreme** point in direction **d** if it maximizes the dot product **p** • **d**



Definition: given a vector **d**, a point **p** is an **extreme** point in direction **d** if it maximizes the dot product **p • d**



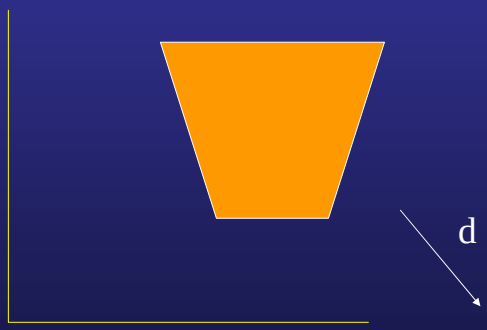
กำหนดให้ $|\mathbf{d}|=1$



Definition: given a vector **d**, a point **p** is an **extreme** point in direction **d** if it maximizes the dot product **p • d**

Definition: given a vector **d**, a point **p** is an **extreme** point in direction **d** if it maximizes the dot product **p • d**

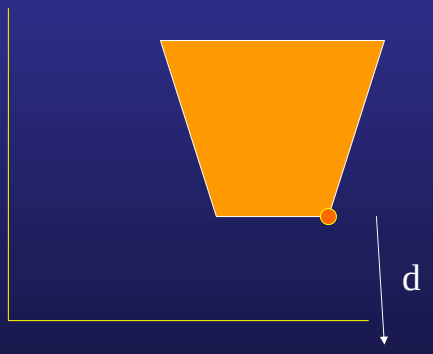
Definition: given a vector **d**, a point **p** is an **extreme** point in direction **d** if it maximizes the dot product **p • d**



Definition: given a vector **d**, a point **p** is an **extreme** point in direction **d** if it maximizes the dot product **p • d**

Definition: given a vector **d**, a point **p** is an **extreme** point in direction **d** if it maximizes the dot product **p • d**

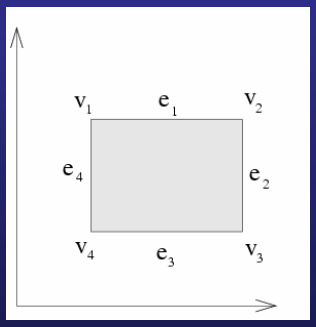
Definition: given a vector **d**, a point **p** is an **extreme** point in direction **d** if it maximizes the dot product **p • d**



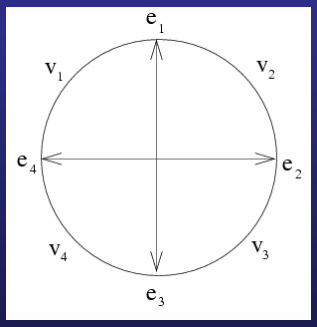
Definition: given a vector **d**, a point **p** is an **extreme** point in direction **d** if it maximizes the dot product **p • d**

Definition: given a vector **d**, a point **p** is an **extreme** point in direction **d** if it maximizes the dot product **p • d**

จุดสุดขีด (extreme point) ของรูป *P* ในทิศทาง **d** ก็คือจุดของรูป *P* ที่อยู่ที่ไกลสุดในทิศทาง **d**



รูป *P*



แสดงจุดสุดขีดของ *P* ในทิศทางต่างๆ

Observation: given two convex polygon *P* and *R*, the set of extreme points of $P \oplus R$ in direction **d** is the set of sums of points **p** in *P* and points **r** in *R* that are extreme in direction **d**.

True because of linearity of the dot product, namely,
$(\mathbf{p}+\mathbf{r})\bullet\mathbf{d} = \mathbf{p}\bullet\mathbf{d}+\mathbf{r}\bullet\mathbf{d}$

That is, let

P_d and R_d be the set of extreme points of P and R in direction d.

The set of extreme points in direction d of $P \oplus R$ is

$$P_d \oplus R_d$$

หมายความว่า

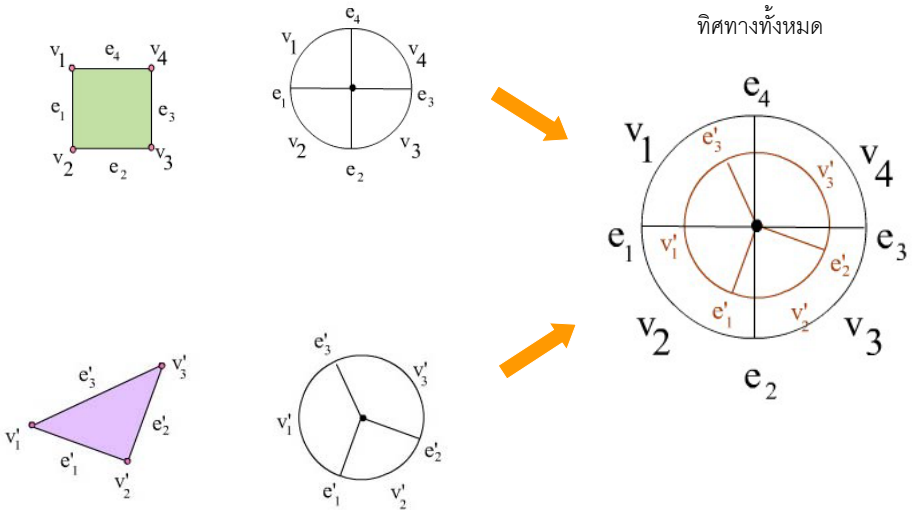
ที่ทิศทาง d ใดๆ ถ้าจุด c เป็นจุดสุดขีดของ $P \oplus R$ แล้วละก็ c เป็นผลบวกของจุดสุดสุดขีดในทิศทาง d ของ P และ R

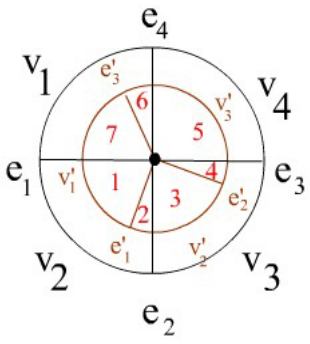
พิสูจน์ เราสามารถเขียนได้ว่า $\mathbf{c} = \mathbf{a} + \mathbf{b}$ สำหรับ $\mathbf{a} \in A$ และ $\mathbf{b} \in B$ ความสมมาตรของ $\mathbf{c} = \mathbf{a} + \mathbf{b}$ ทำให้เพียงพอที่จะสมมุติขัดแย้งว่า a ไม่ใช่เวกเตอร์สุดขีดของ A ในทิศทาง d นั่นคือมี $\mathbf{a}' \in A$ ซึ่ง $\mathbf{a}' \bullet \mathbf{d} > \mathbf{a} \bullet \mathbf{d}$

เมื่อพิจารณา $\mathbf{c}' = \mathbf{a}' + \mathbf{b}$ ด้วยคุณสมบัติการกระจายของผลคูณภายใน เราได้ว่า $\mathbf{c}' \bullet \mathbf{d} = \mathbf{a}' \bullet \mathbf{d} + \mathbf{b} \bullet \mathbf{d} > \mathbf{c} \bullet \mathbf{d}$ ทำให้ c ไม่ใช่เวกเตอร์สุดขีดในทิศทาง d ขัดแย้งกับที่กำหนดไว้ □

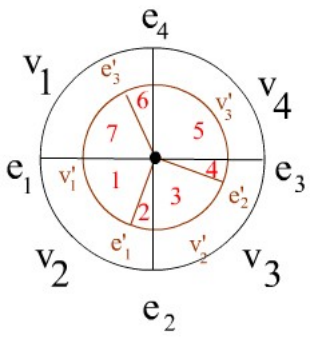
เมื่อ P และ R เป็นรูปหลายเหลี่ยมคอนเวกซ์ เรารู้ว่า $P \oplus R$ ก็เป็นรูปหลายเหลี่ยมคอนเวกซ์ด้วย ซึ่งจุดสุดขีดของ $P \oplus R$ ในทิศทาง d ใดๆ ก็เป็นผลบวกของจุดสุดขีดของ P และ R ในทิศทางนั้น

ดังนั้นเราสามารถคำนวณ $P \oplus R$ ได้โดยหาจุดสุดขีดในทุกๆ ทิศทางโดยบวกจุดสุดขีดของ P และ R ในแต่ละทิศทาง





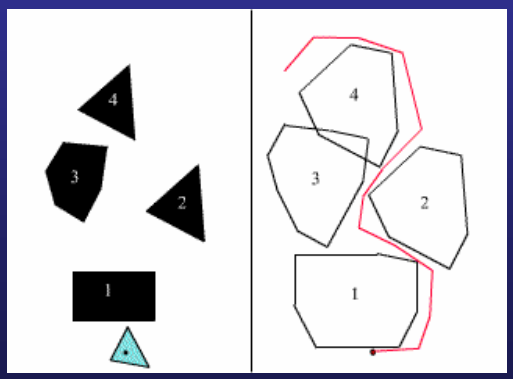
$$\mathbf{p}_1 = \mathbf{v}'_1 + \mathbf{v}_2$$
$$\mathbf{p}_2 = \mathbf{v}'_2 + \mathbf{v}_2$$
$$\mathbf{p}_3 = \mathbf{v}'_2 + \mathbf{v}_3$$
$$\mathbf{p}_4 = \mathbf{v}'_3 + \mathbf{v}_3$$
$$\mathbf{p}_5 = \mathbf{v}'_3 + \mathbf{v}_4$$
$$\mathbf{p}_6 = \mathbf{v}'_3 + \mathbf{v}_1$$
$$\mathbf{p}_7 = \mathbf{v}'_1 + \mathbf{v}_1$$



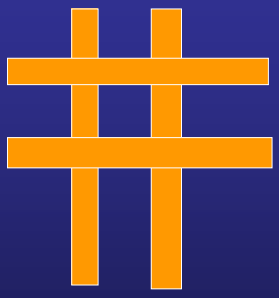
Given two convex polygon P and R with m and n edges, their minkowski sum $P \oplus R$ can be computed in O(m+n) time and consists at most m+n edges.

แต่เรามีผลรวมมิงคอฟสกีหลายชิ้นมายูเนียนกัน

ยูเนียนของมันจะมีจำนวนเส้นขอบเท่าไร



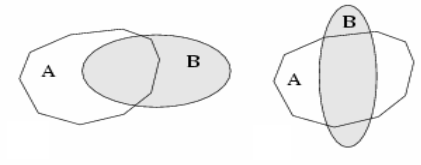
จะเป็นอย่างนี้ได้ไหม? $O(n^2)$



ไม่มีทางเพราะ C-Obstacle ที่ได้จากผลรวมมิงคอฟสกีไม่ทับกันอย่างนั้น

เพราะมันเป็นแผ่นกลมเทียม (pseudodisk)

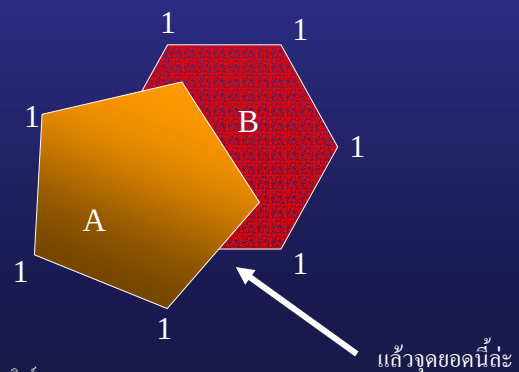
A และ B เป็นแผ่นกลมเทียมถ้า A-B และ B-A เป็นชิ้นเดียว



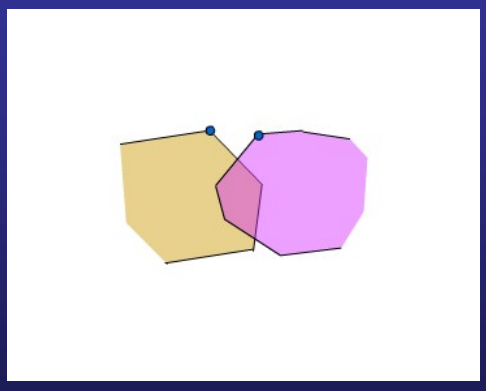
เป็น ไม่เป็น

แล้วยูเนียนของแผ่นกลมเทียมที่เป็นรูปหลายเหลี่ยมจะมีจำนวนเส้นขอบเท่าไร ?

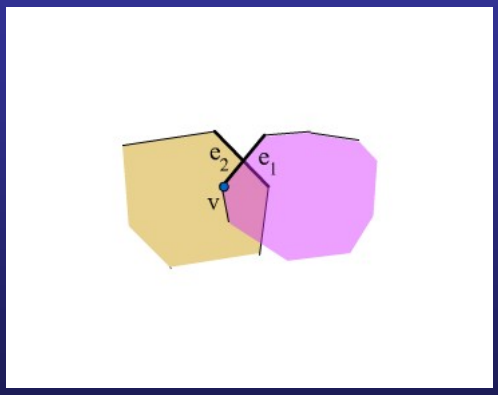
ก็ลองนับจุดยอดของยูเนียนดู โดยให้นับว่าทุกจุดยอดรอบยูเนียนมีประจุจุดละหนึ่ง แต่ละประจุต้องเอาไปฝาก โดยฝากได้เฉพาะที่จุดยอดของ A และ B เท่านั้น จุดยอดของ A และ B ที่เป็นจุดยอดของยูเนียนก็ฝากไว้ที่ตัวมันเอง แล้วจุดยอดของยูเนียนที่เกิดจากการตัดกันล่ะ



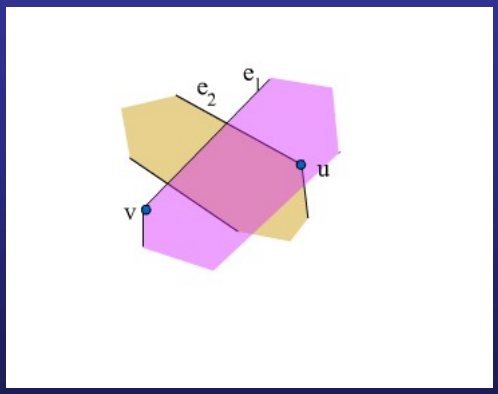
พิจารณาการตัดกันของเส้นขอบ



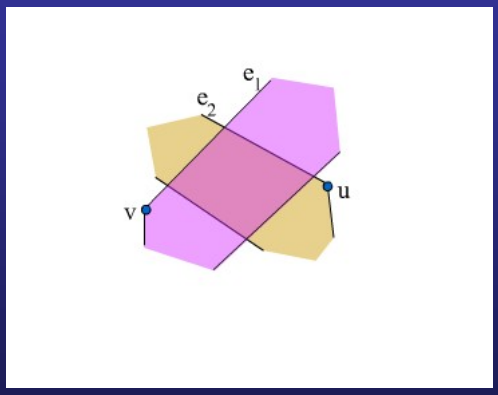
ถ้าตัดกันแบบนี้โดยมีจุดปลาย e_1 อยู่ภายในก็ให้นำประจุของจุดตัดไปฝากไว้ได้



แต่ถ้าจุดปลาย e_1 ทะลุผ่านให้พิจารณาจุดปลายของ e_2 ถ้ามันอยู่ภายใน ก็ให้ฝากได้



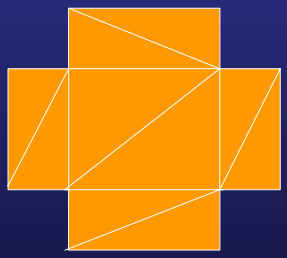
แต่ถ้าไม่จะเห็นได้ว่าแต่ละรูปถูกอีกรูปหนึ่งตัดผ่าน ซึ่งเป็นไปไม่ได้ถ้ามันเป็นแผ่นกลมเทียม



So, the number of vertices in the union of a collection of pseudodisks with a total of n vertices is at most 2n = O(n).

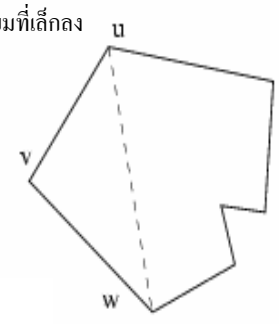
แล้วในกรณีที่สิ่งกีดขวางไม่เป็นคอนเวกซ์ล่ะ

เราก็สามารถแบ่งมันเป็นสามเหลี่ยมแล้วหา C-Obstacle จากนั้นนำผลลัพธ์มายูเนียนกัน

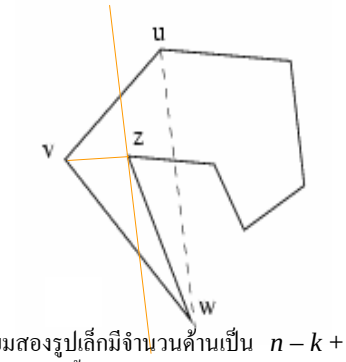


สิ่งกีดขวางรูป n เหลี่ยมที่ไม่เป็นคอนเวกซ์สามารถแบ่งเป็นสามเหลี่ยม n-2 รูปได้เสมอ

พิสูจน์ พิจารณาจุดยอดที่อยู่ซ้ายสุด หากมีมากกว่าหนึ่งให้เลือกจุดที่ต่ำสุด ให้ชื่อว่า v และเรียกจุดยอดทั้งสองที่ติดกับมันว่า u และ w พิจารณาสามเหลี่ยม uvw ถ้าไม่จุดยอดอื่นในสามเหลี่ยมนี้ เราสามารถลากเส้น uw เพื่อแบ่งเป็นรูปเหลี่ยมเป็นรูปสามเหลี่ยมกับรูปเหลี่ยมที่เล็กลง



แต่ถ้ามีจุดอื่นอยู่ในสามเหลี่ยม ให้เลือกจุดที่อยู่ไกลจากเส้น uw มากที่สุด เรียกมันว่า z (เมื่อพิจารณาเส้นที่ขนานกับ uw และผ่านจุด z จะได้ว่าไม่มีเส้นขอบใดอยู่ทางซ้ายของเส้นนี้และอยู่ภายในสามเหลี่ยม uvw) ดังนั้นเราสามารถลากเส้น vz โดยไม่ตัดกับเส้นขอบใด นั่นก็คือเราสามารถแบ่งรูปเหลี่ยมที่ให้ออกเป็นรูปเหลี่ยมที่เล็กลง



จะเห็นได้ว่ารูปหลายเหลี่ยมสองรูปเล็กมีจำนวนด้านเป็น $n-k+1$ และ $k+1$ ซึ่งตามสมมุติฐานจะมีจำนวนสามเหลี่ยมทั้งหมด $[(n-k+1)-2]+[(k+1)-2]=n-2$ □

Lemma 1: given a set of convex objects $T_1, T_2, \ldots, T_n$ with disjoint interior and a convex object R. The set

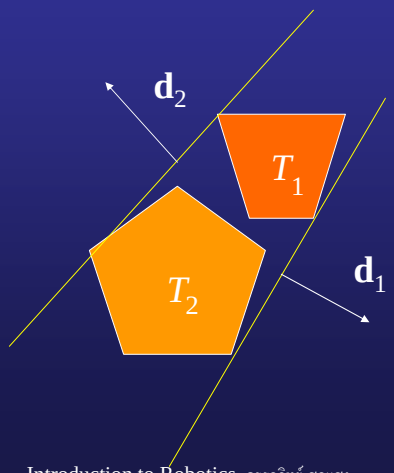
$$\{T_i \oplus R \mid 1 \le i \le n\}$$

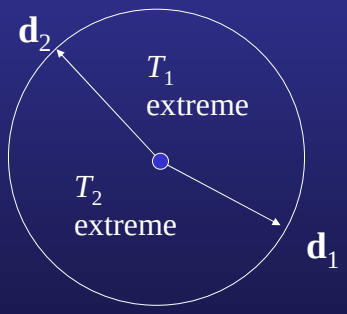
is a collection of pseudodisks.

Proof: Consider two convex polygons T_1 and T_2 with disjoint interior, we want to show that $T_1 \oplus R$ and $T_2 \oplus R$ do not cross over one another.

Recall the condition for a point to be extreme in a given direction **d**…

Observation For T_1 and T_2, we have a double tangents. Let $\mathbf{d}_1$ and $\mathbf{d}_2$ be the outward pointing perpendicular vectors of these tangents.



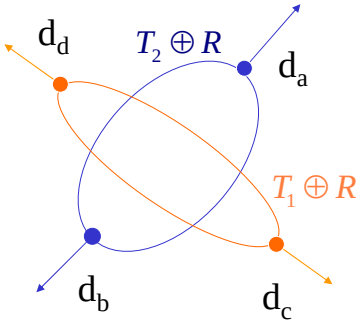


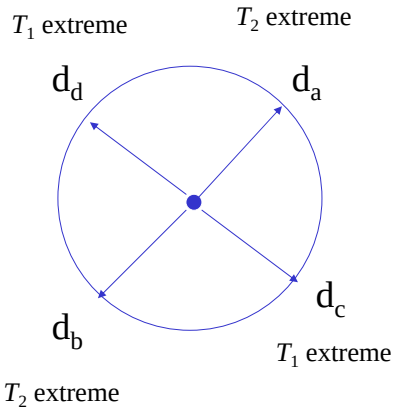
p is an extreme point of $T_1 \oplus R$ in direction d, then there exist:

t_1 = an extreme point of T_1 in direction d
r = an extreme point of R in direction d

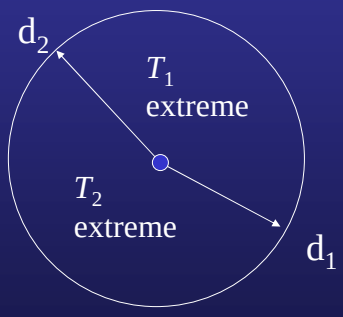
Such that $p = t_1 + r$

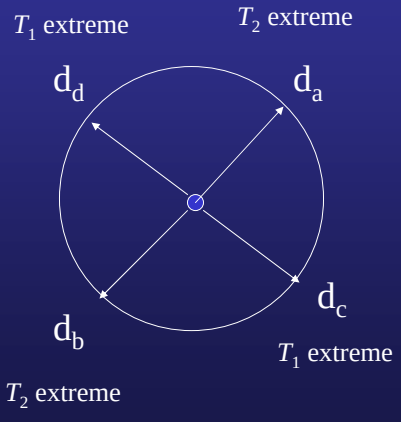
Suppose the Lemma isn't true. That is there exist T_1 and T_2 such that $T_1 \oplus R$ and $T_2 \oplus R$ cross over.





But wait… this is not possible. Recall from the observation how the extremity switches.





Therefore,

Lemma 1: given a set of convex objects $T_1, T_2, \ldots, T_n$ with disjoint interior and a convex object R. The set

$$\{T_i \oplus R \mid 1 \le i \le n\}$$

is a collection of pseudodisks.