Spatial Data Structures

COLLEGE OF COMPUTING

HANYANG ERICA CAMPUS

Q YOUN HONG (홍규연)

Data Structures for Computer Graphics

사실적인 그래픽스 화면을 렌더링하기 위해 필요한 요소:

- 더 많은 프레임(frame)
- 더 높은 해상도 + 샘플링 비율
- 더 사실적인 조명 표현
- 더 복잡한 기하 모델, 등

⇒ 이를 위해 가속화 알고리즘 + 효율적인 **자료 구조** 필요

Ray tracing으로 렌더링한 Boeing 777 모델:
(132,500개의 part, 3M 개의 나사로 구성되어 있으며) 350M 개 이상의 삼각형을 렌더링함

Data Structures for Computer Graphics

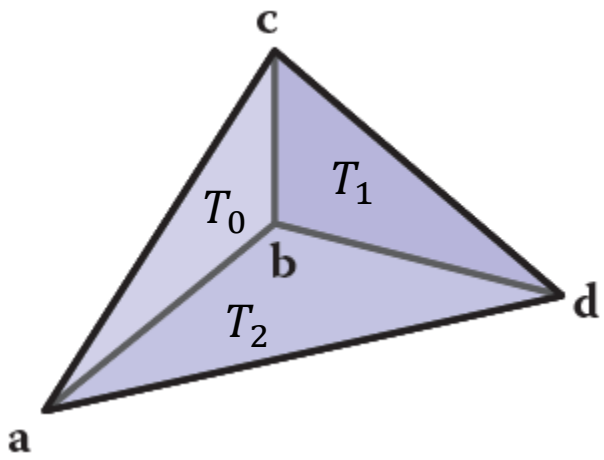
컴퓨터 그래픽스에서 사용하는 자료 구조들:

- 메시 (Mesh structures (revisited))
- Scene Graph
- 공간 자료 구조 (Spatial Data Structures)
  - Object partitioning: ex) bounding volume hierarchy (BVH)
  - Space partitioning: ex) binary space partitioning (BSP)

Mesh Structure (revisited)

Simple mesh representation:

separated triangles vs. indexed triangle mesh



Triangle Id	Vertex 0	Vertex 1	Vertex 2
0	(a_x, a_y, a_z)	(b_x, b_y, b_z)	(c_x, c_y, c_z)
1	(b_x, b_y, b_z)	(d_x, d_y, d_z)	(c_x, c_y, c_z)
2	(a_x, a_y, a_z)	(d_x, d_y, d_z)	(b_x, b_y, b_z)

[Method 1] Separated triangles

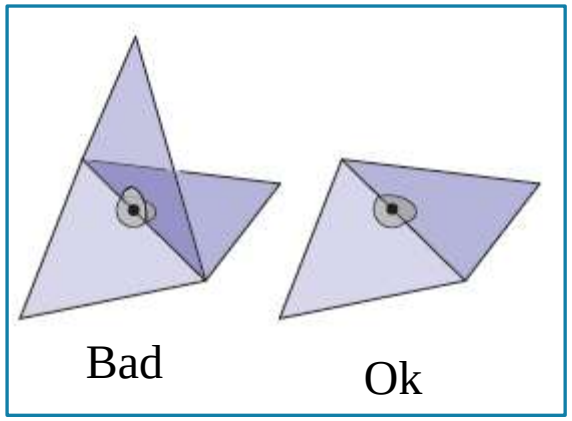
Triangle Id	Vertex Ids
0	(0,1,2)
1	(1,3,2)
2	(0,3,1)

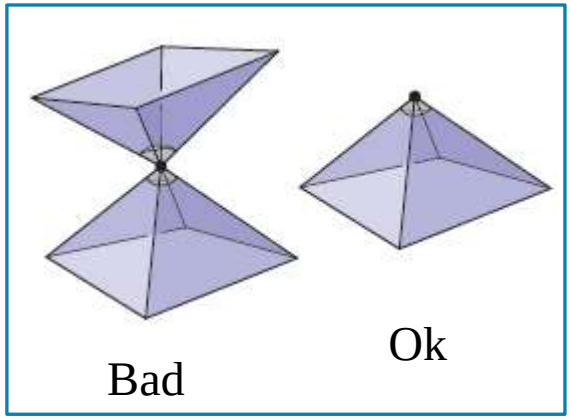
Vertex Id	Vertex Ids
0	(a_x, a_y, a_z)
1	(b_x, b_y, b_z)
2	(c_x, c_y, c_z)
3	(d_x, d_y, d_z)

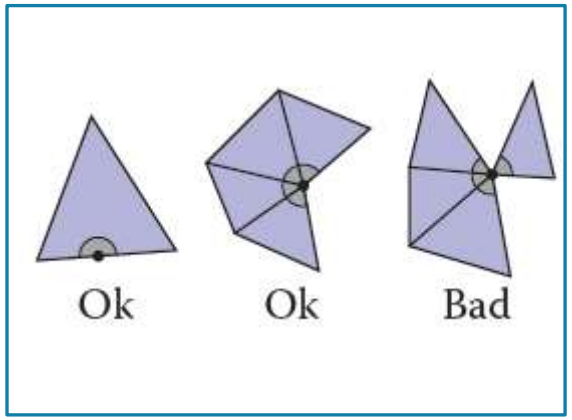
[Method 2] Shared vertices

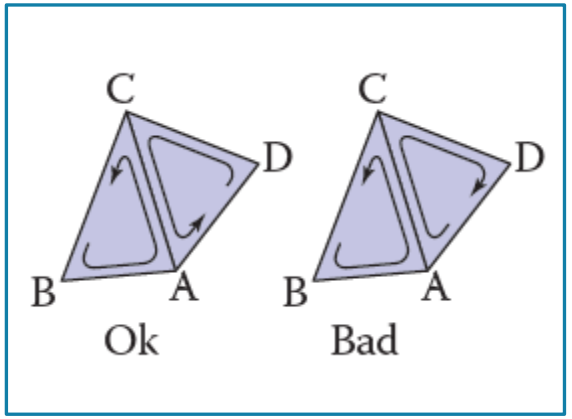
(Triangular) Mesh Structure

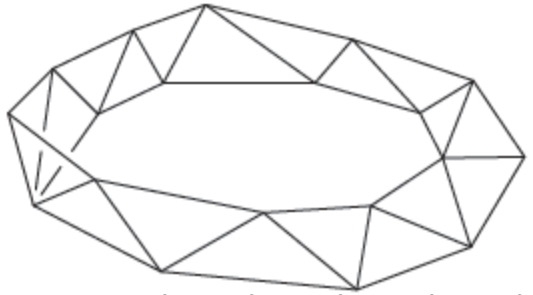
- Mesh must be manifold
  - 모든 edge는 두 triangle에 의해 공유되어야 함
  - 모든 vertex는 그 vertex를 공유하는 하나의 (닫힌) loop이 있어야 함
    - 예외: mesh에 boundary를 허용할 경우 열린 loop
- Mesh must have consistent orientation











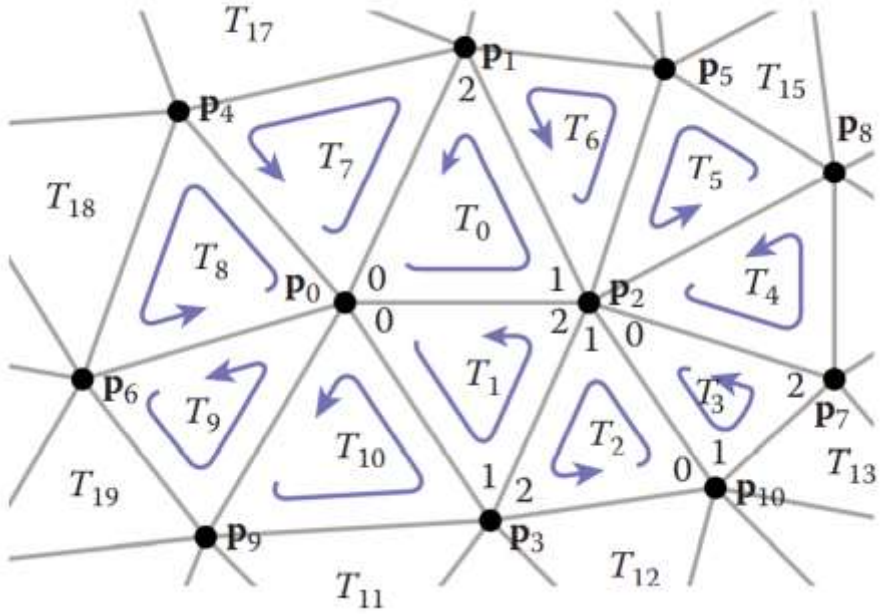
Triangulated Möbius band

Mesh Structure

Q) Mesh 구조에 대해 다음과 같은 query의 답을 어떻게 계산하는가?

- Mesh 상의 어떤 삼각형 **t**와 이웃하는 세 삼각형은?
- Mesh 상의 어떤 edge **e**에 대해 **e**를 공유하는 두 삼각형은?
- Mesh 상의 어떤 vertex **v**를 포함하는 모든 edge (또는 삼각형)은?

⇒ Mesh connectivity information (adjacency information) needed!



Mesh Structure: 가장 단순한 경우


```
Triangle {
    Vertex v[3];
    Edge e[3];
}

Edge {
    Vertex v[2];
    Triangle t[2];
}

Vertex {
  //per-vertex data
  Triangle t[]
  Edge e[]
}
```

① Mesh 상의 어떤 삼각형 **t**와 이웃하는 세 삼각형은?
② Mesh 상의 어떤 edge **e**에 대해 **e**를 공유하는 두 삼각형은?
③ Mesh 상의 어떤 vertex **v**를 포함하는 모든 edge (또는 삼각형)은?

위의 질문들에 답하기 위해 각 vertex **v**에 그 vertex를 포함하는 모든 삼각형들과 edge들 저장

⇒ 질문 ③ 의 답을 constant time에 계산할 수 있음
⇒ Vertex의 크기(in memory)가 가변적
⇒ 질문 ①,②의 답은?

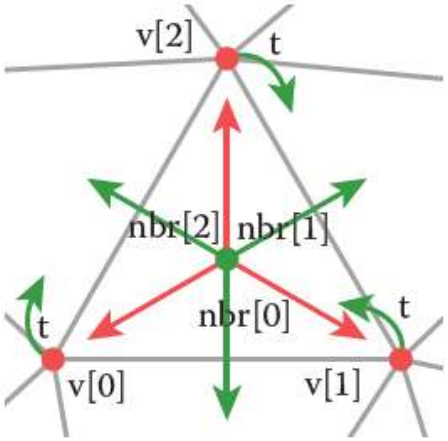
Mesh Structure: Triangle-Neighbor Structure

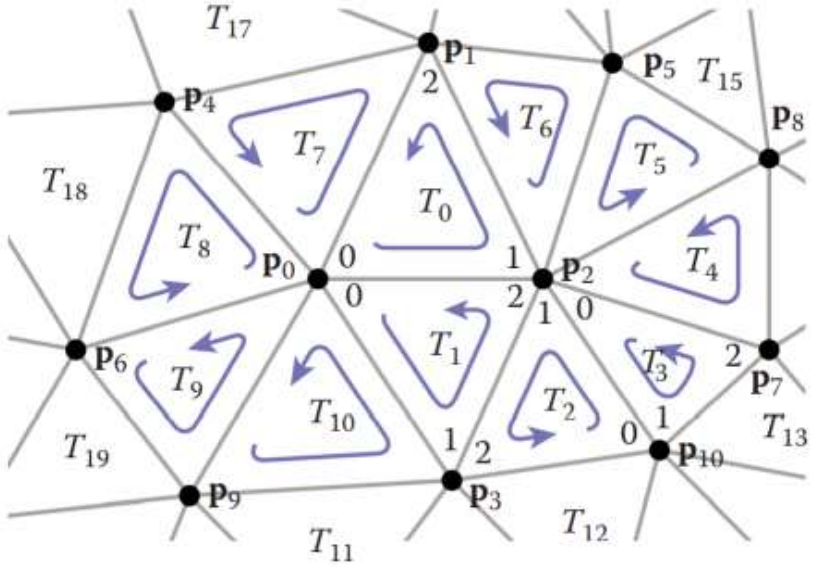

```
Triangle {
  Triangle nbr[3];
  Vertex v[3];
}

Vertex {
  //any adjacent tri
  Triangle t;
```

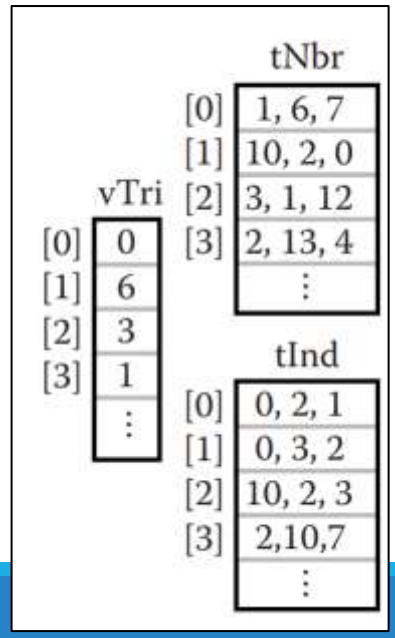
Or

```
Mesh {
  //vertex indices in triangles
  int tInd[nt][3];
  //indices of neighbor triangles
  int tNbr[nt][3];
  //indices of any adjacent traignels
  int vTri[nv];
}
```

- Triangle.nbr[k]: Edge Triangle.v[k]와 Triangle.v[(k+1)%3]를 공유하는 삼각형의 index(Id)
- Vertex.t: 이 vertex를 포함하는 아무 삼각형







Mesh Structure: Triangle-Neighbor Structure


```
Triangle {
  Triangle nbr[3];
  Vertex v[3];
}

Vertex {
  //any adjacent tri
  Triangle t;
```

Q) Mesh 상의 어떤 vertex **v**를 포함하는 모든 삼각형은?

```
TrianglesOfVertex(v) {
  t = v.t
  do {
    find i such that (t.v[i] == v)
    t = t.nbr[i]
  } while (t != v.t)
}
```

⇒ i 를 구하기 위해 매번 삼각형의 모든 vertex를 체크해야 함

⇒ Solution: triangle에서 이웃하는 triangle들을 저장하는 대신 이웃 triangle들의 한 edge를 저장

Mesh Structure: Triangle-Neighbor Structure


```
Triangle {
  Triangle nbr[3];
  Vertex v[3];
}

Vertex {
  //any adjacent tri
  Triangle t;
```

```
Triangle {
  Edge nbr[3];
  Vertex v[3];
}
Edge {
  Triangle t;
  int i; // in {0, 1, 2}
}
Vertex {
  //any edge leaving vertex
  Edge e;
```

Q) Mesh 상의 어떤 vertex **v**를 포함하는 모든 삼각형은?

```
TrianglesOfVertex(v) {
  t = v.t
  do {
    find i such that (t.v[i] == v)
    t = t.nbr[i]
  } while (t != v.t)
}
```

```
TrianglesOfVertex(v) {
  {t, i} = v.e
  do {
    {t, i} = t.nbr[i]
    i = (i + 1) % 3
  } while (t != v.e.t)
}
```

삼각형의 모든 vertex를 체크해야

이웃하는 triangle들을 저장하는 대

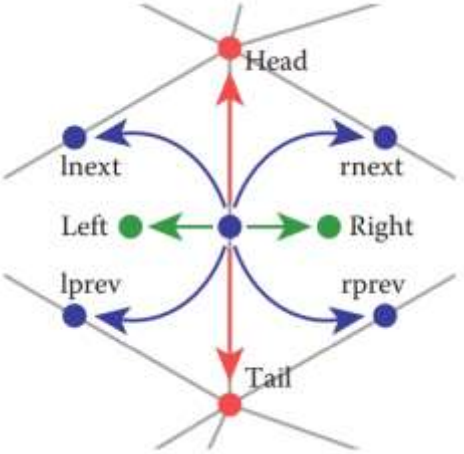
edge를 저장

Mesh Structure: Winged-Edge Structure


```
Edge {
  Edge lprev, lnext,
       rprev, rnext;
  Vertex head, tail;
  Face left, right;
}
Face {
  //...any face info
  //any adjacent edge
  Edge e;
}
Vertex {
  //...any vertex info
  //any incident edge
  Edge e;
}
```

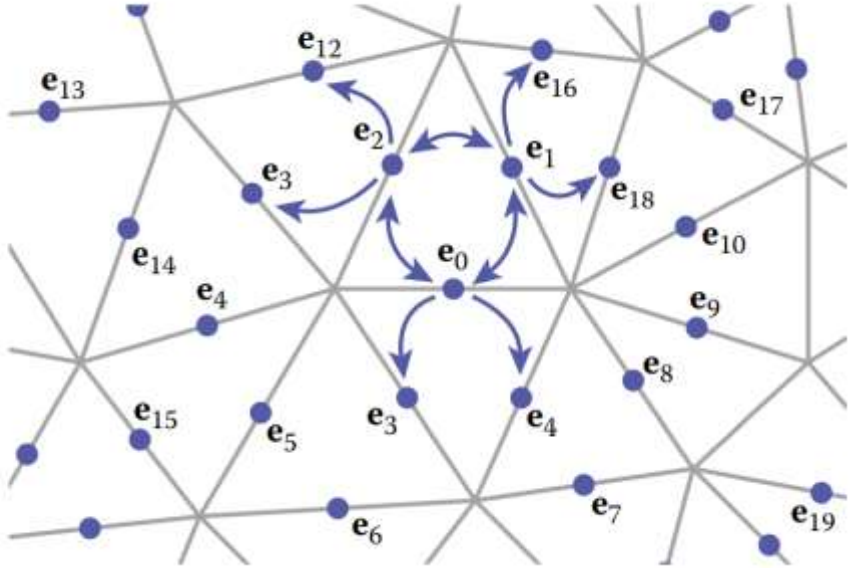
- 각 Edge에 connectivity 정보를 저장:

- head, tail: 이 edge의 머리/꼬리 vertex
- left, right: 이 edge를 포함하는 왼쪽, 오른쪽 삼각형
- lprev, lnext: left 삼각형에서 봤을 때 이 edge의 앞/뒤 edge
- rprev, rnext: right 삼각형에서 봤을 때 이 edge의 앞/뒤 edge



Winged-edge table

	ln	rn	lp	rp
[0]	1	4	2	3
[1]	18	0	16	2
[2]	12	1	3	0
	⋮			

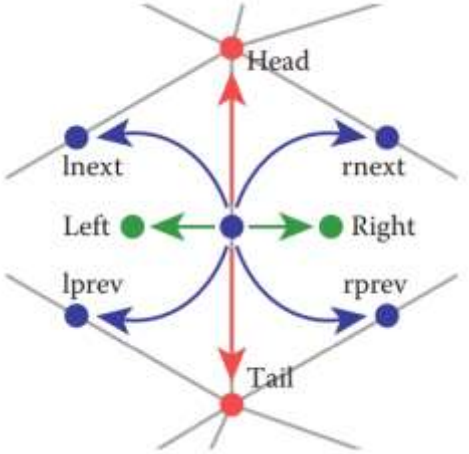


Mesh Structure: Winged-Edge Structure


```
Edge {
  Edge lprev, lnext,
       rprev, rnext;
  Vertex head, tail;
  Face left, right;
}
Face {
  //...any face info
  //any adjacent edge
  Edge e;
}
Vertex {
  //...any vertex info
  //any incident edge
  Edge e;
}
```

Q) Mesh 상의 어떤 vertex **v**를 포함하는 모든 edge은?

```
EdgesOfVertex(v) {
  e = v.e;
  do {
    if (e.tail == v)
      e = e.lprev;
    else
      e = e.rpnext;
} while (e != v.e)
```

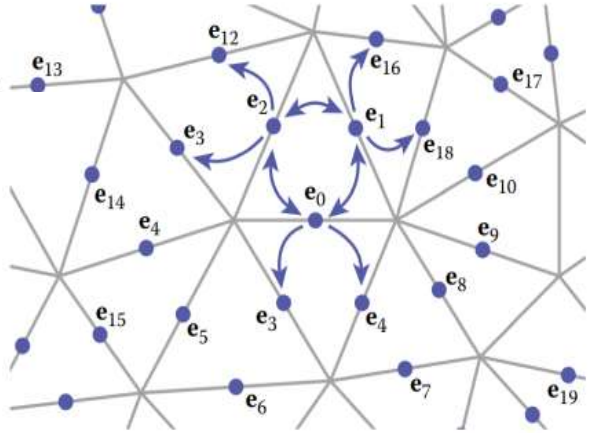


Q) 한 triangle에 속한 edge 구하기

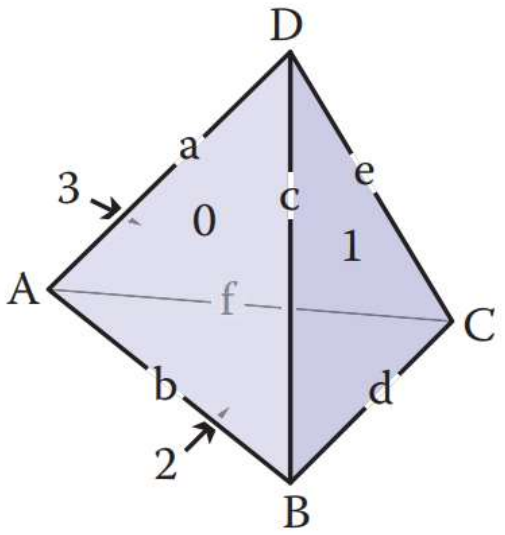
```
EdgesOfFace(f) {
  e = f.e;
  do {
    if (e.left == f)
      e = e.lnext;
    else
      e = e.rprev;
  } while (e != f.e);
}
```

Winged-edge table

	ln	rn	lp	rp
[0]	1	4	2	3
[1]	18	0	16	2
[2]	12	1	3	0
	⋮			



Mesh Structure: Winged-Edge Structure



Edge	*Vertex 1*	*Vertex 2*	*Face left*	*Face right*	*Pred left*	*Succ left*	*Succ right*	*Pred right*
a	A	D	3	0	f	e	c	b
b	A	B	0	2	a	c	d	f
c	B	D	0	1	b	a	e	d
d	B	C	1	2	c	e	f	b
e	C	D	1	3	d	c	a	f
f	C	A	3	2	e	e	b	d

Vertex	*Edge*
A	a
B	d
C	d
D	e

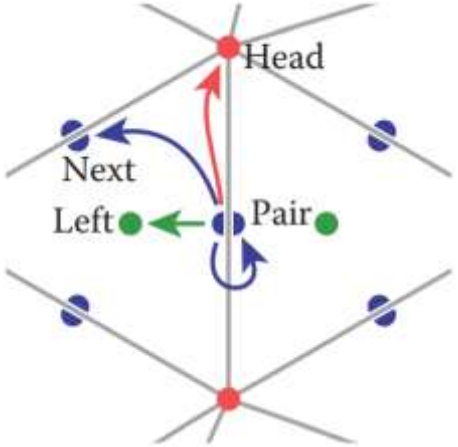
Face	*Edge*
0	a
1	c
2	d
3	a

Mesh Structure: Half-Edge Structure

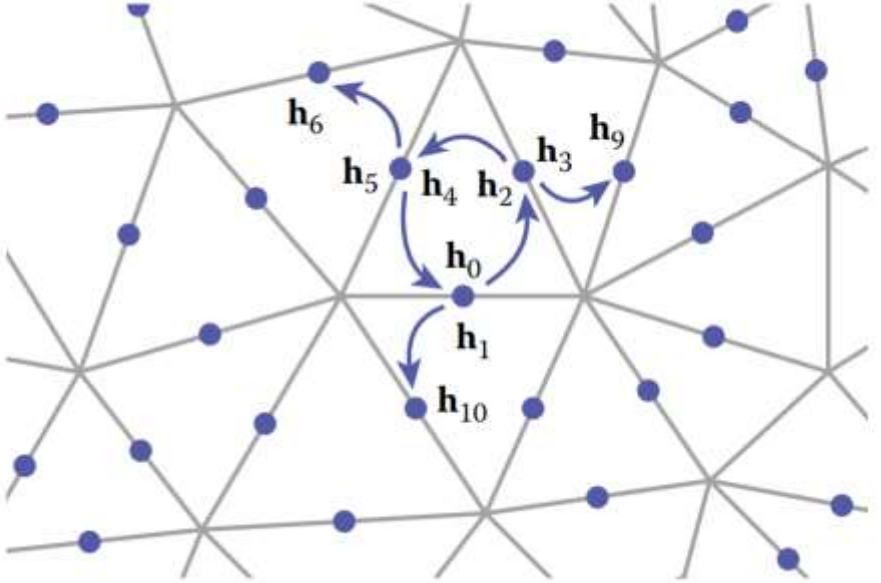

```
HEdge {
  HEdge pair, next;
  Vertex v;
  Face f;
}
Face {
  //...any face info
  //any h-edge of this face
  HEdge h;
}
Vertex {
  //...any vertex info
  //any outgoing h-edge
  Hedge h;
}
```

- Winged-edge를 두 단방향 edge로 나눔

- HEdge.v: 이 half-edge가 향하는 vertex
- HEdge.pair: half-edge의 반대 방향 half-edge
- HEdge.next: half-edge를 포함하는 삼각형 상에서 이 half-edge의 다음 half-edge



	Pair	Next
hedge[0]	1	2
hedge[1]	0	10
hedge[2]	3	4
hedge[3]	2	9
hedge[4]	5	0
hedge[5]	4	6
	⋮	

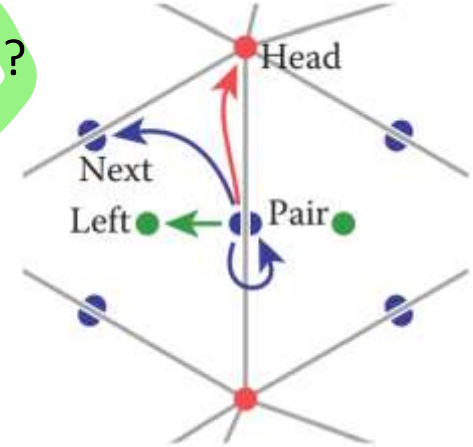


Mesh Structure: Half-Edge Structure


```
HEdge {
  HEdge pair, next;
  Vertex v;
  Face f;
}
Face {
  //...any face info
  //any h-edge of this face
  HEdge h;
}
Vertex {
  //...any vertex info
  //any outgoing h-edge
  HEdge h;
}
```

Q) Mesh 상의 어떤 vertex **v**를 포함하는 모든 edge은?

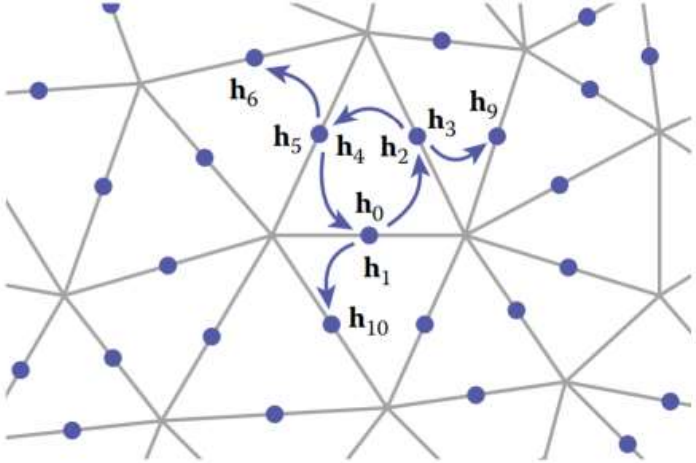
```
EdgesOfVertex(v) {
  h = v.h;
  do {
    h = h.pair.next;
  } while (h != v.h)
}
```



Q) 한 triangle에 속한 edge 구하기

```
EdgesOfFace(f) {
  h = f.h;
  do {
    h = h.next;
  } while (h != f.h);
}
```

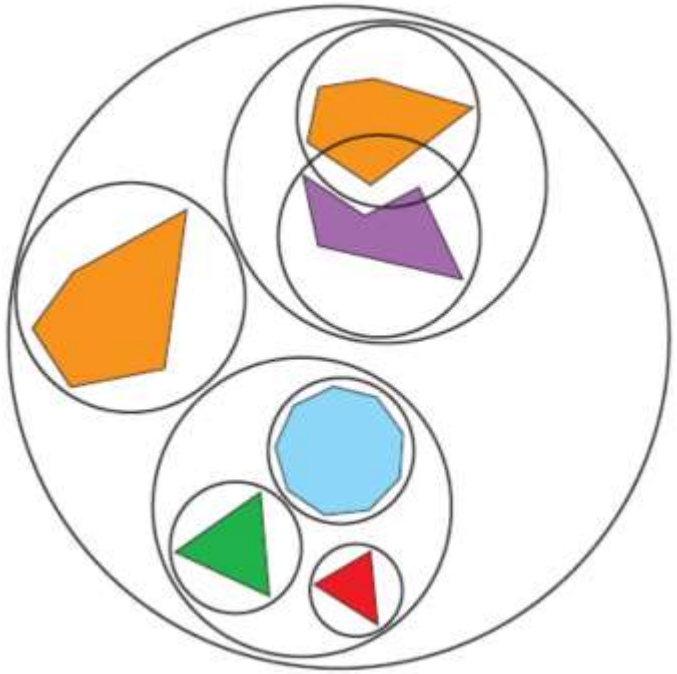
	Pair	Next
hedge[0]	1	2
hedge[1]	0	10
hedge[2]	3	4
hedge[3]	2	9
hedge[4]	5	0
hedge[5]	4	6
	⋮	

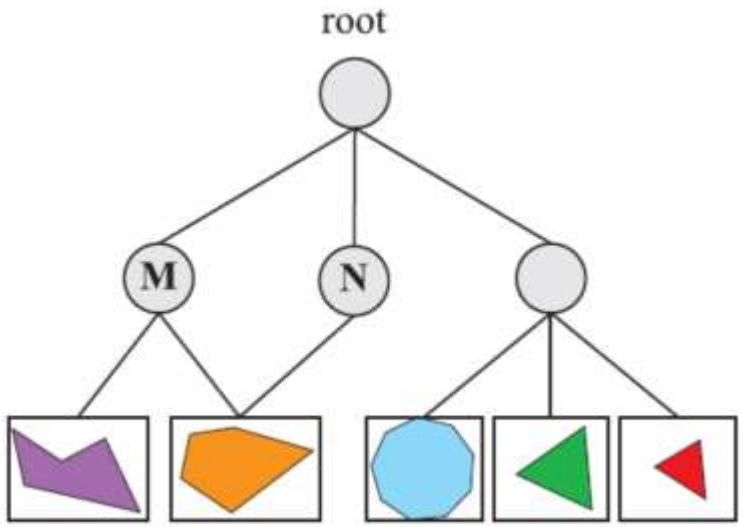


Scene Graph

복잡한 Scene에서 여러 물체의 위치를 계층적(Hierarchical) 구조로 표현하는 방법

- 물체에 적용되는 transformation을 tree 구조로 저장
- Leaf에 있는 물체의 위치는 tree에서 ancestor에 해당하는 모든 transformation을 곱하여 구함 => matrix stack 이용

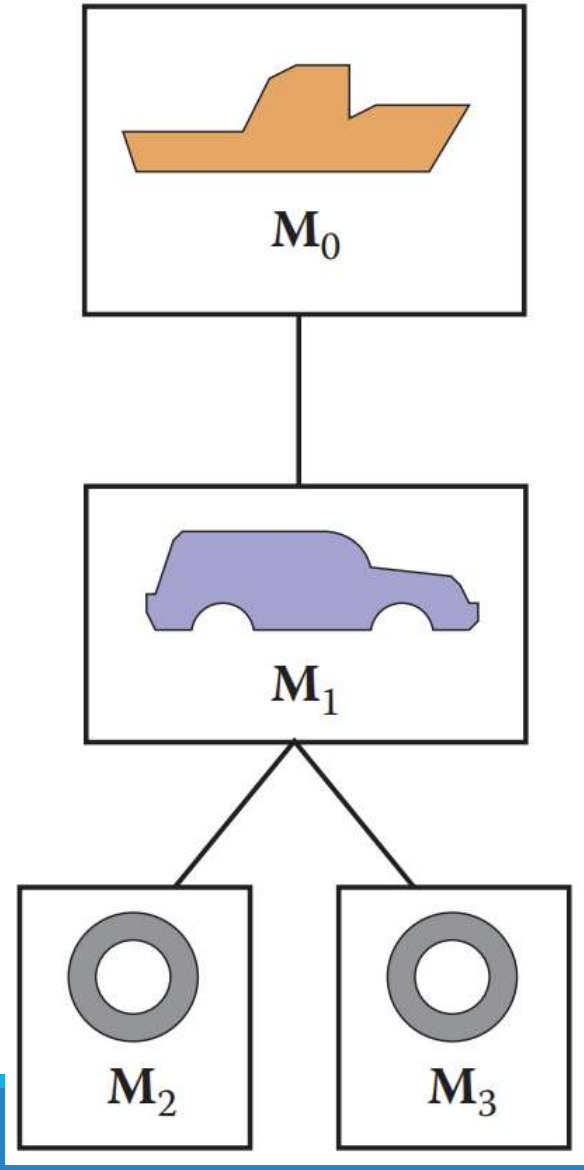




Scene Graph: 예시

- Ferry transformation M_0
- Car transformation M_0M_1
- Left wheel transformation $M_0M_1M_2$
- Right wheel transformation $M_0M_1M_3$

```
Function traverse(node)
  push(M_local) // push transformation matrix into a stack
  draw object using composite matrix from stack
  traverse(left child)
  traverse(right child)
  pop // pop a matrix from a matric stack
```



Spatial Data Structure

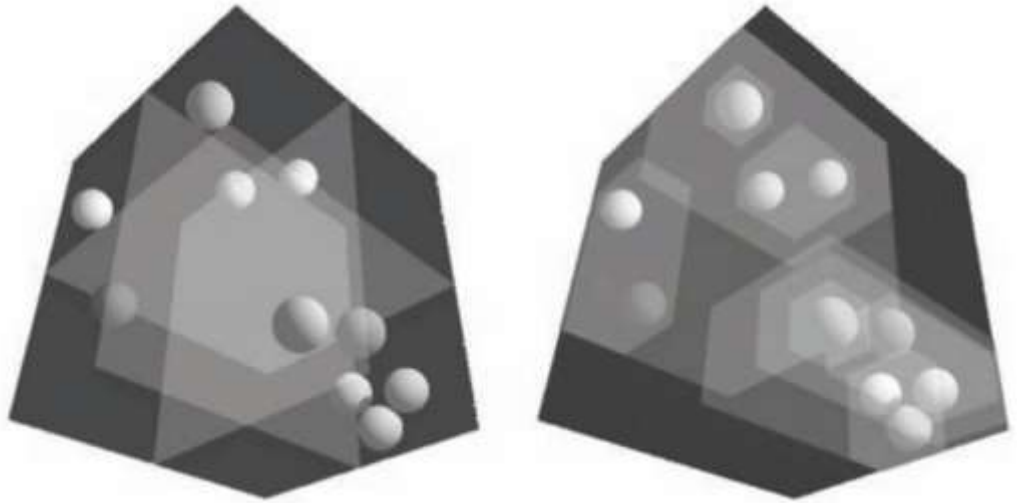
Spatial data structure (공간자료구조)의 중요성:

- 그래픽스 연산의 가속화하기 위해 geometric object를 공간상에서 효율적으로 관리하는 자료 구조 필요
- 예시)

- In ray tracing: ray가 object와 충돌하는지 빠르게 테스트
- In computer games, physical simulation: scene에 있는 object들이 서로 충돌하는지 감지
- In culling: viewing frustum안에 있지 않은 object들을 제거하여 화면에 그려지지 않게 함

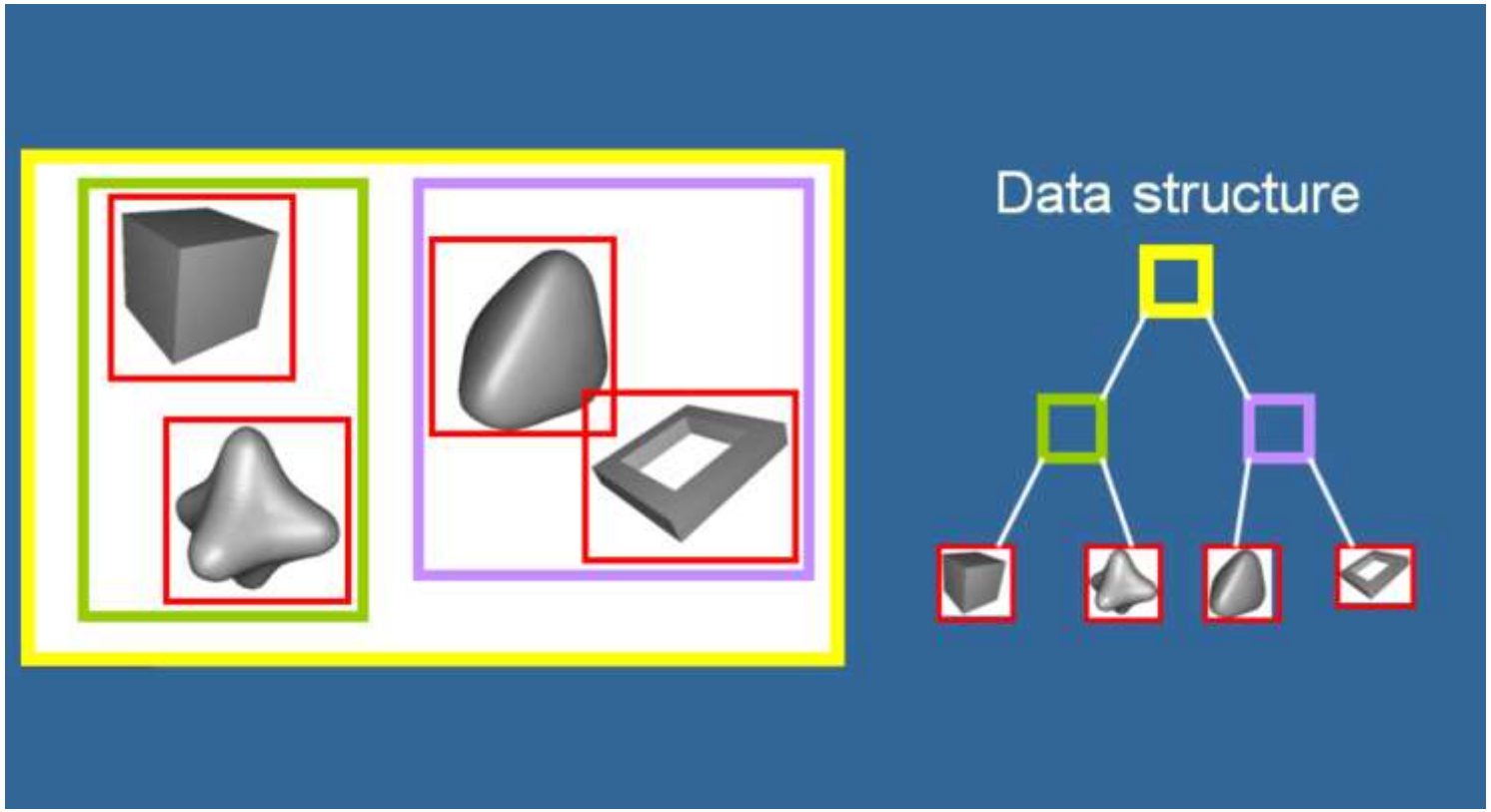
Spatial Data Structure의 종류

- Object partitioning method
  - 물체를 (겹치지 않는) 그룹으로 나누고, 그룹의 계층(hierarchy)를 구서
  - 각 그룹은 공간상으로는 겹칠 수 있음
- Space partitioning method
  - 공간을 계층적으로 나누고, 나눈 공간에 속하는 물체들을 묶음
  - 한 물체가 여러 partition에 속할 수 있음



(왼쪽) space partitioning의 예 (uniform space partitioning)
(오른쪽) object partitioning의 예 (bounding boxes)

Object Partitioning: Bounding Volume Hierarchy



- 각 물체를 보다 단순한 물체로 완전히 감쌈
  예) 박스 (axis-aligned box, oriented box), 구, 사면체 등

- Hierarchy의 root부터 tree를 traverse하며 기하 query (i.e. 충돌 감지)를 실행:
  - If answer = No (i.e. no collision): 그 node의 전체 subtree를 skip
  - If answer = Yes (i.e. maybe collision): 그 node의 child node에 대해 재귀적으로 query를 실행

- Bounding volume들을 최대한 tight하게 정의해야 더 많이 가속화됨

Bounding Volume Hierarchy: (Axis-aligned Bounding Box)

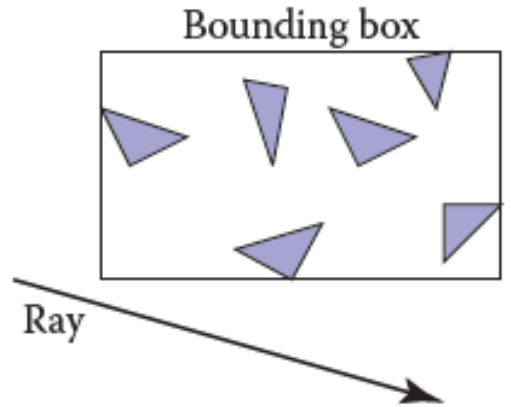
- 2D Bounding box – 다음의 4 선분으로 둘러싸인 2D 영역
  - $x = x_{min},\ x = x_{max},\ y = y_{min},\ y = y_{max}$
- Q) 주어진 ray와 a bounding box가 충돌하는가?

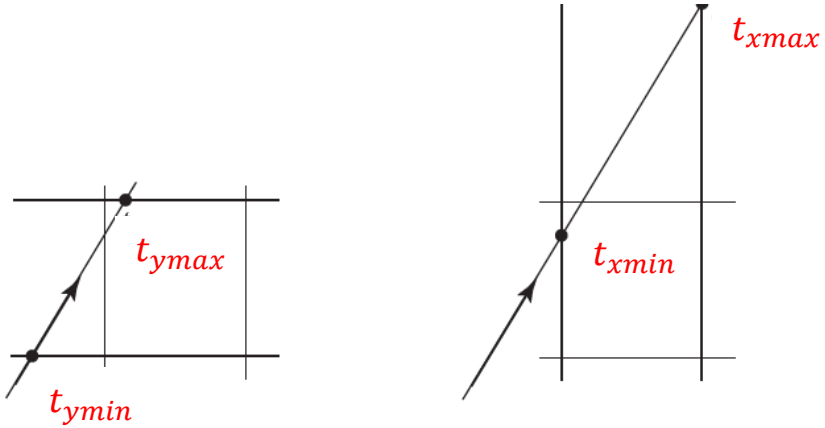
⇒ intersection intervals 검사

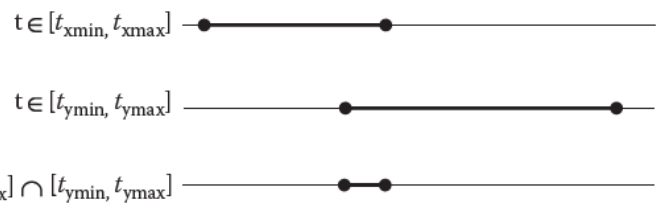
- Ray 상의 한 점의 식: $p(t) = (x_e, y_e) + (x_d, y_d)t$
- Finding t's that intersect $x = x_{min},\ x = x_{max}$

  $t_{xmin} = \frac{x_{min} - x_e}{x_d},\quad t_{xmax} = \frac{x_{max} - x_e}{x_d}$

  => 만약 $t \in [t_{xmin}, t_{xmax}]$, 이 점은 두 수직선 사이에 위치
- 같은 식으로 만약 $t \in [t_{ymin}, t_{ymax}]$, 이 점은 두 수평선 사이에 위치
- $t \in [t_{xmin}, t_{xmax}], t \in [t_{ymin}, t_{ymax}]$를 동시에 만족할 때 이 점은 2D bounding box 안에 위치

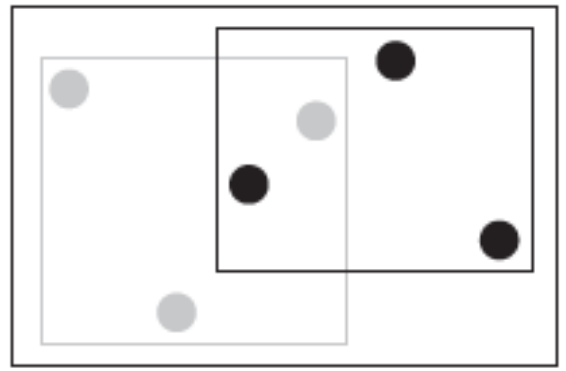




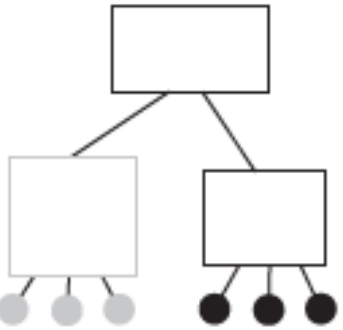


Bounding Volume Hierarchy: (Axis-aligned Bounding Box)

- Hierarchical bounding box – bounding box들의 hierarchy
  - 하나의 bounding box는 그 안에 있는 모든 object들을 항상 bounding
  - Bounding box는 공간 상에서 항상 그 안에 있는 object들을 포함하는 것은 아님



- k-ary tree를 이용해서 표현 (보통 binary tree)



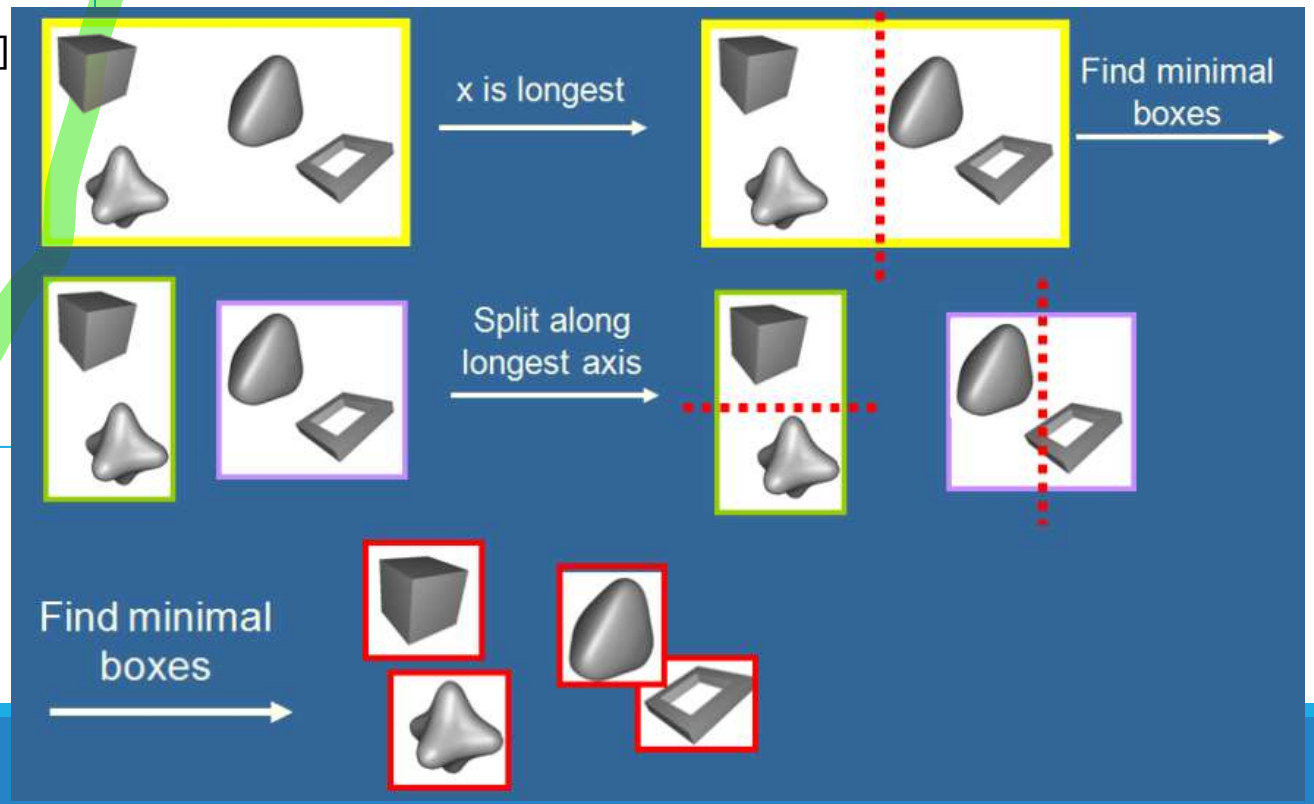
```
BvhNode {
  surface-pointer left;
  surface-pointer right;
  Box bbox;

  virtual box bounding-box()
  virtual bool hit(ray a + tb,
           real t0, real t1,
           hit-record rec);
}
```

```
Function bool BvhNode::hit(ray a + tb,
                  real t0, real t1, hit-record rec){
  if (bbox.hitbox(a + tb, t0, t1) then
    hit-record lrec, rrec;
    left-hit = (left ≠ NULL) and
                  (left→hit(a + tb, t0, t1, lrec))
    right-hit = (right ≠ NULL) and
                  (right→hit(a + tb, t0, t1, rrec))
    if (left-hit and right-hit) then
      rec = (lrec.t < rrec.t) ? lrec : rrec;
      return true;
    else if (left-hit) rec = lrec; return true;
    else if (right-hit) rec= rrec; return true;
    else false;
  else
    return false;
}
```

Bounding Volume Hierarchy: (Axis-aligned Bounding Box)

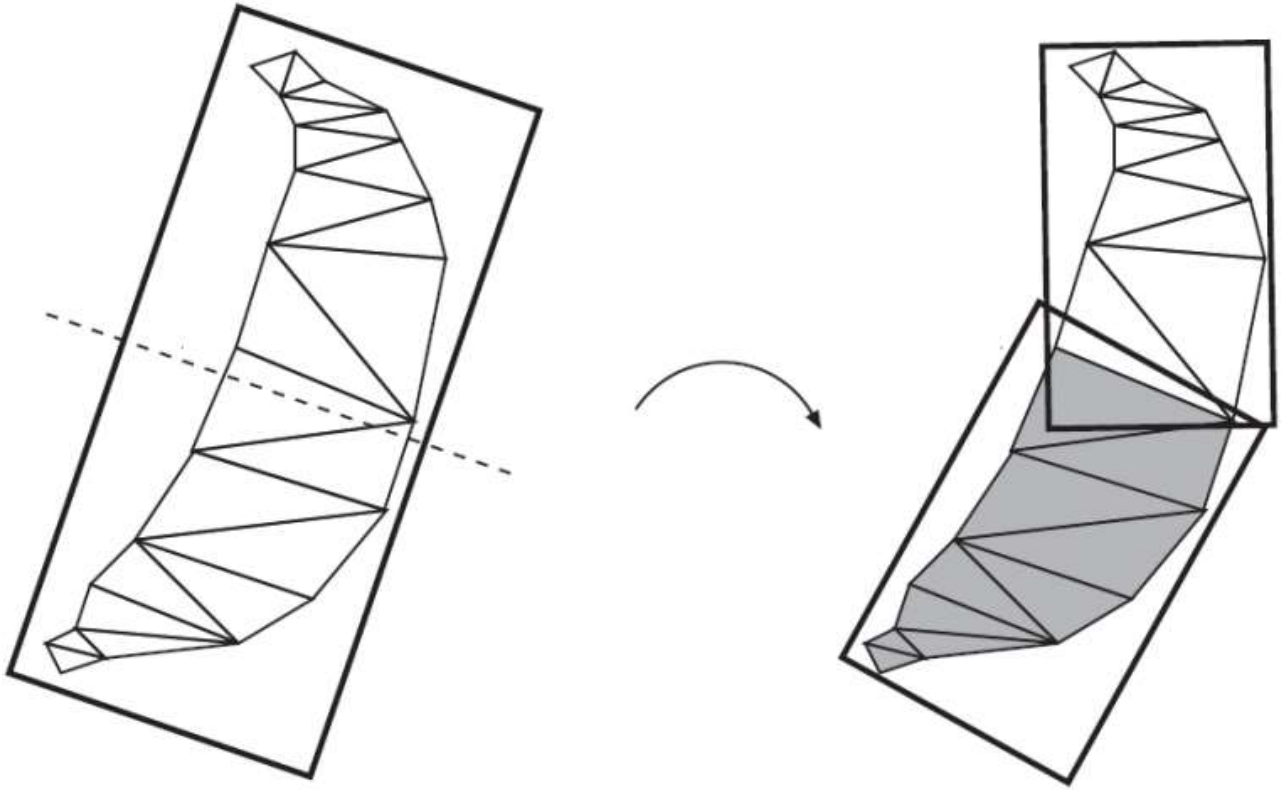
Hierarchical Bounding Box Construction

```
Function BvhNode::create(object-array A, int AXIS){
  N = A.length;
  if (N = 1) then
    left = A[0]; right = NULL;
    bbox = bounding-box(A[0])
  else if (N = 2) then
    left = A[0]; right = A[1];
    bbox = combine(bounding-box(A[0]), bounding-box(A[1]
  else
    find the midpoint m of bounding box of A along AXIS
    partition A into lists with length=k and (N-k)
    find the longest axis NEW_AXIS of bounding box of A
    left = new BvhNode(A[0..k], NEW_AXIS);
    right = new BvhNode(A[k+1,..N-1], NEW_AXIS);
    bbox = combine(left→bbox, right →bbox)
}
```



Bounding Volume Hierarchy: OBB

OBB (Oriented Bounding Box) Construction

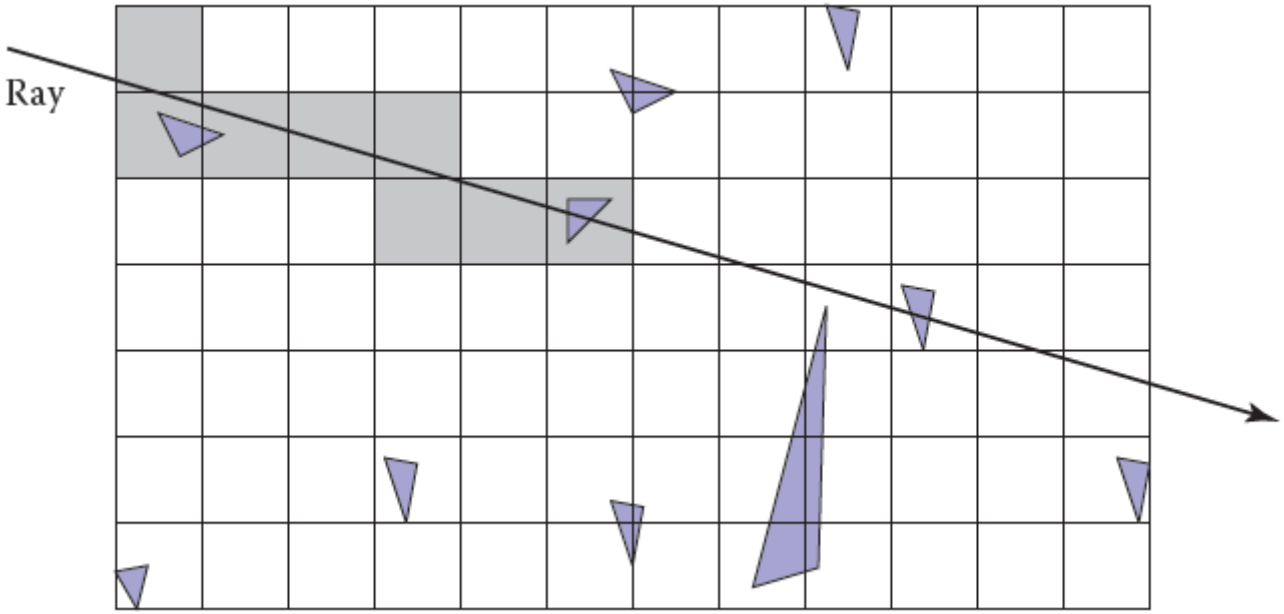


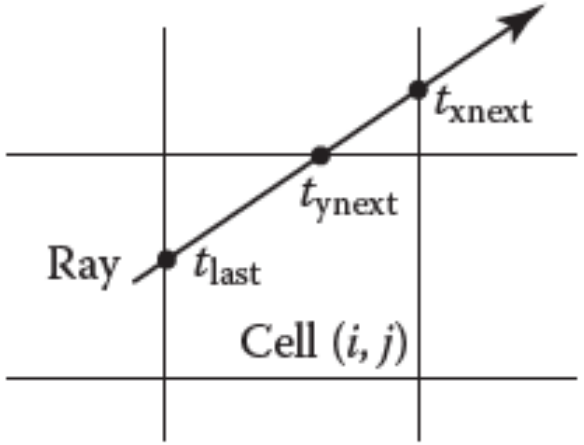
Uniform Spatial Partitioning

한양대학교 ERICA
소프트웨어융합대학
COLLEGE OF COMPUTING

하나의 Scene은 동일한 크기의 grid로 나누어짐

- 하나의 물체이 여러 cell에 위치할 수 있음
- Ray intersection problem: ray-cell intersection를 먼저 확인하고, 충돌이 있는 cell 안에 있는 물체들만 체크





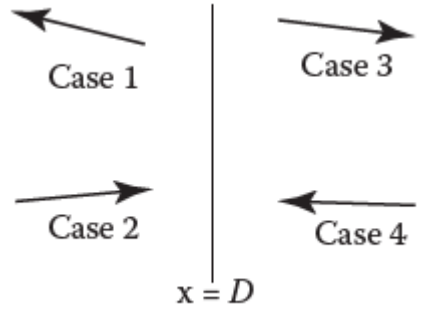
Binary Space Partitioning (BSP)

하나의 Scene을 cutting plane을 이용해서 left subtree와 right subtree로 나눔

⇒ subtree들을 새로운 cutting plane을 이용해서 재귀적으로 나눔

```
BspNode {
  surface-pointer left;
  surface-pointer right;
  Box bbox;
  real D; //cutting plane x=D
  virtual box bounding-box()
  virtual bool hit(ray a + tb,
           real t0, real t1,
           hit-record rec);
}
```

```
Function bool BvhNode::hit(ray a + tb,
                  real t0, real t1, hit-record rec){
  xp = xa + t0xb;
  if (xp < D) then
    if (xb < 0) then
      return (left ≠ NULL) and
             (left→hit(a + tb, t0, t1, rec))
    t = (D − xa)/xb
    if (t < t1) then
      return (left ≠ NULL) and
             (left→hit(a + tb, t0, t1, rec))
    if (left ≠ NULL) and (left→hit(a + tb, t0, t1, rec)) then
      return true;
    return (right ≠ NULL) and (right→hit(a + tb, t, t1, rec))
  else
    similary for case 3 and 4…;
}
```



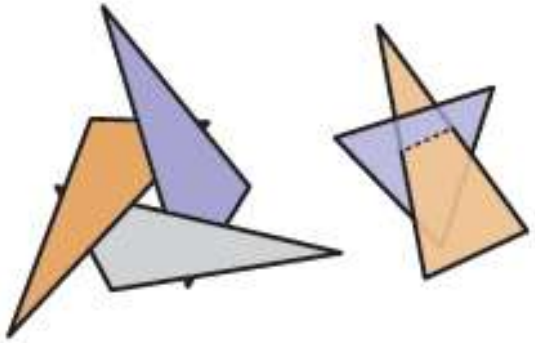
BSP Trees for Visibility

BSp Tree algorithm:

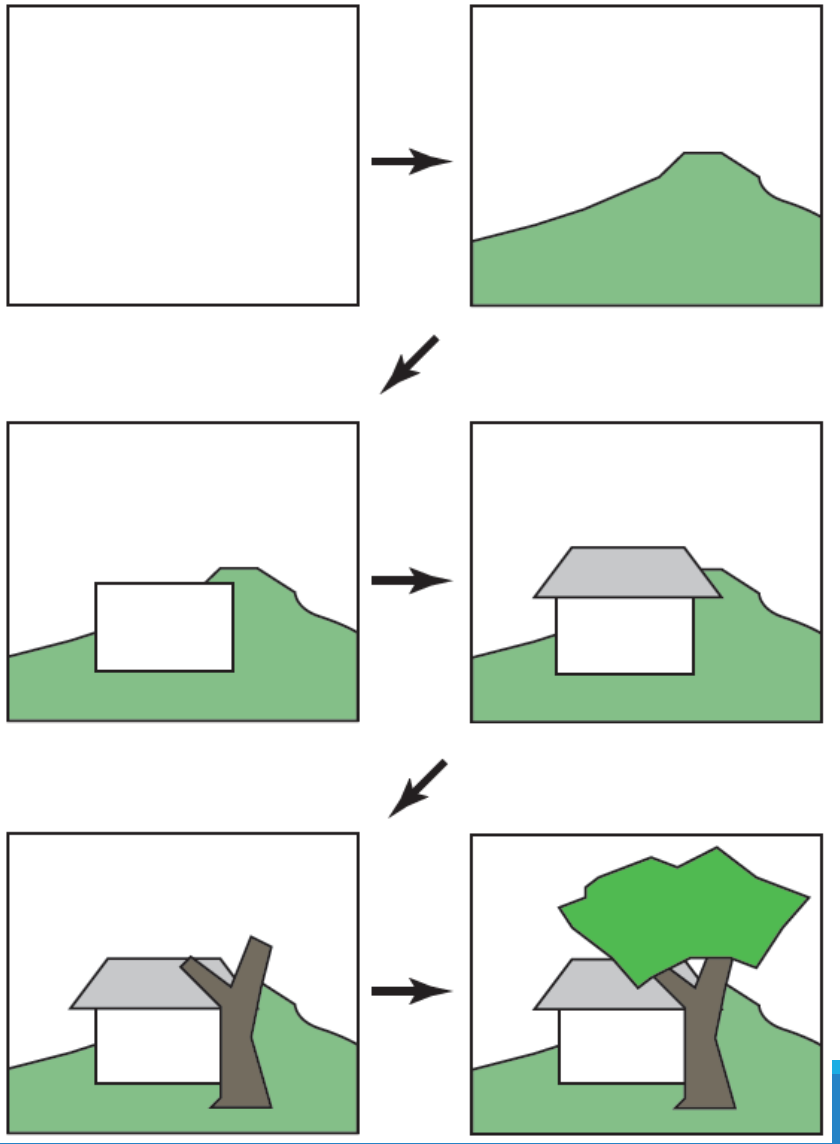
- painter's algorithm (물체를 뒤에서부터 앞의 순서로 그림)

1. Sort objects back to front relative to viewpoint
2. For each object, draw object on screen

- 자연적으로 hidden surface들을 제거함
- 기본 알고리즘: global back-to-front order 존재한다고 가정 (global back-to-front order이 불가능할 경우, 별도의 전처리 과정 필요)



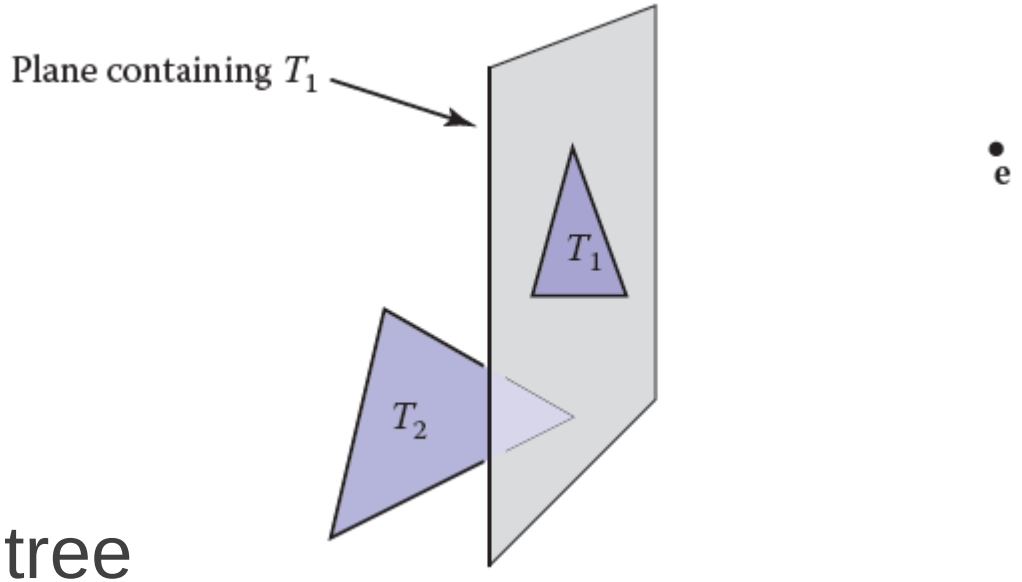
Global back-to-front ordering not possible



BSP Tree for Visibility

- Draw objects back-to-front:

```
if (f1(e) < 0) then
    draw T1
    draw T2
else
    draw T2
    draw T1
```



- Drawing many objects using BSP tree

```
Function draw(bsp tree, point e)
  if tree.empty then return
  if (f_tree.root(e) < 0) then
   draw (tree.plus, e)
   rasterize tree.triangle
   draw (tree.minus, e)
  else
   draw (tree.minus, e)
   rasterize tree.triangle
   draw (tree.plus, e)
```

BSP Tree for Visibility