컴퓨터 그래픽스
제5장 2차원 그래픽스의 변환

2019년 2학기

5장 학습 내용

- **2차원 그래픽스 변환**
  - 기본 변환: 이동, 회전, 신축
  - 그 외, 반사, 밀림
  - 동차 좌표계
  - 윈도우와 뷰포트
  - 클리핑 알고리즘

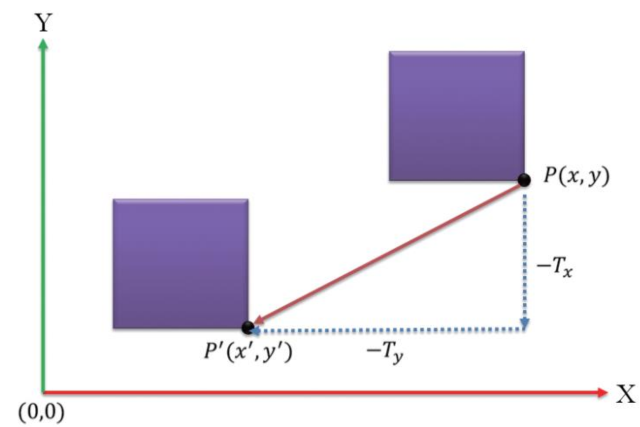
기본 기하 변환: 이동

- **기본 기하 변환**
  - 이동 (Translation)
  - 회전 (Rotation)
  - 신축 (크기 변환, Scale)

- **Translation (이동)**
  - 좌표계의 한 곳에서 다른 곳으로 직선 경로를 따라 객체의 위치를 바꾸는 것
  - 객체의 크기나 모양, 방향 등은 바뀌지 않는다.
  - P(x, y) → P'(x', y')

    $x' = x+t_x$ $y' = y+t_y$

    (t_x, t_y): 이동 벡터 (translation vector)



행렬을 사용하면

P' =

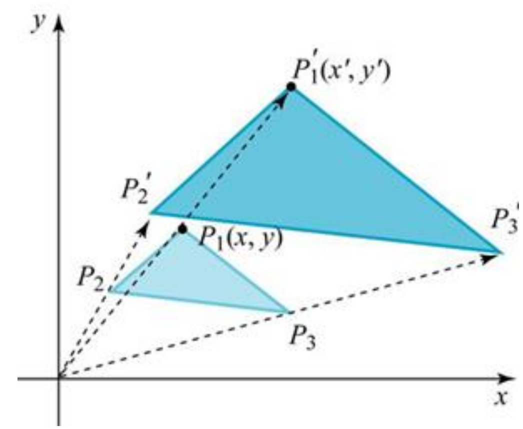
기본 기하 변환: 신축

- Scaling (신축, 확대/축소)
  - 객체의 크기를 확대/축소 시킨다.
  - 객체의 크기뿐 아니라 기준점으로부터의 위치도 배율에 따라 변한다.
  - $P(x, y) \rightarrow P'(x', y')$

    $x' = s_x \bullet x$ $\qquad\qquad$ $y' = s_y \bullet y$

    (s_x, s_y): 신축률 (scaling factor)

행렬을 사용하면



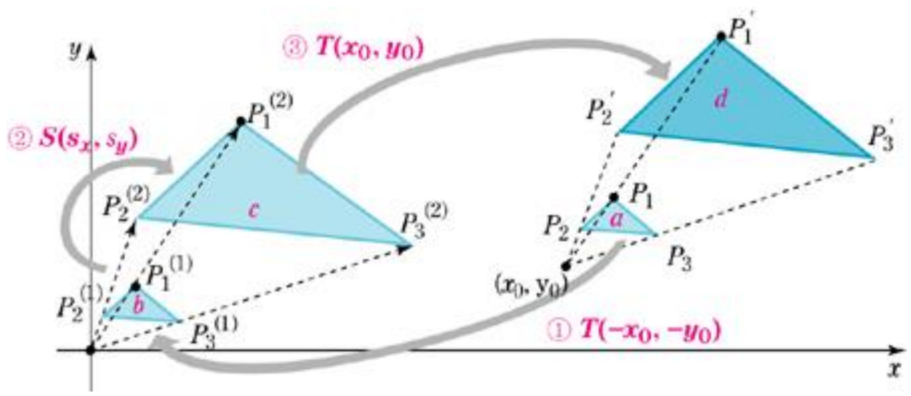
P' =

- s > 1:
- s = 1:
- 0 < s < 1:
- s < 0:

기본 기하 변환: 신축

- 임의의 점 (x_0, y_0)에 대하여 신축률 (s_x, s_y)만큼 신축
  - 신축 기준점을 원점이 되도록 객체를 이동: $T(-x_0, -y_0)$
  - 원점에 대하여 신축: $S(s_x, s_y)$
  - 제자리로 이동: $T(x_0, y_0)$

  - x' =
  - y' =



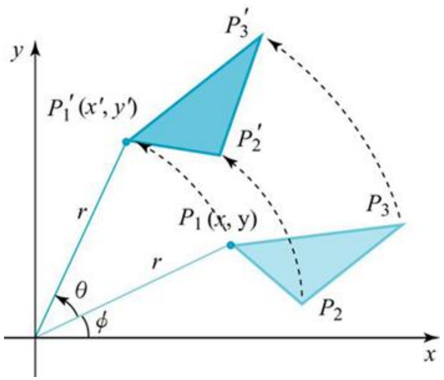
기본 기하 변환: 회전

- Rotation (회전)
  - xy평면에서 원 경로를 따라 객체를 재배치
  - 객체의 모양 변화는 없이 객체가 놓여있는 방향이 변한다.
  - P(x, y) → P'(x', y')
    - $x' = r\cos(\Phi+\theta) = \underline{r\cos\Phi}\cos\theta - \underline{r\sin\Phi}\sin\theta = x\cos\theta - y\sin\theta$
    - $y' = r\sin(\Phi+\theta) = \underline{r\cos\Phi}\sin\theta + \underline{r\sin\Phi}\cos\theta = x\sin\theta + y\cos\theta$

회전각 : θ, 회전점 (Pivot Point): (x_r, y_r)

  - 행렬을 사용하면

P' =

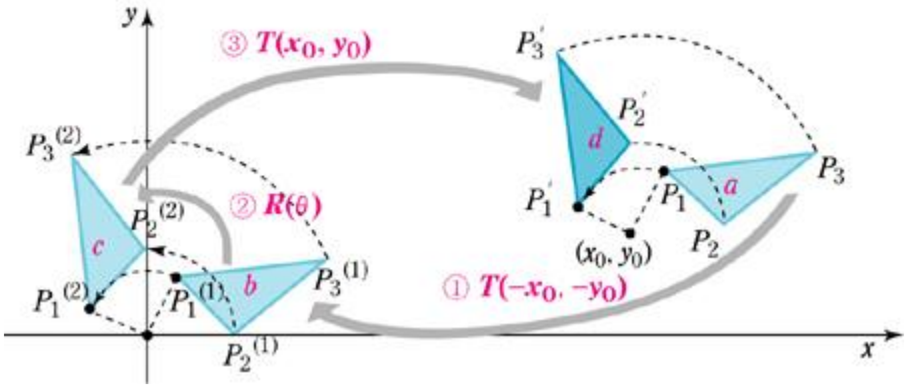


기본 기하 변환: 회전

– 임의의 점 (x_0, y_0)에 대하여 θ만큼 회전
  - 회전 중심점이 원점이 되도록 객체를 이동: $T(-x_0, -y_0)$
  - 원점을 중심으로 θ만큼 회전: $R(\theta)$
  - 반대 방향으로 이동: $T(x_0, y_0)$

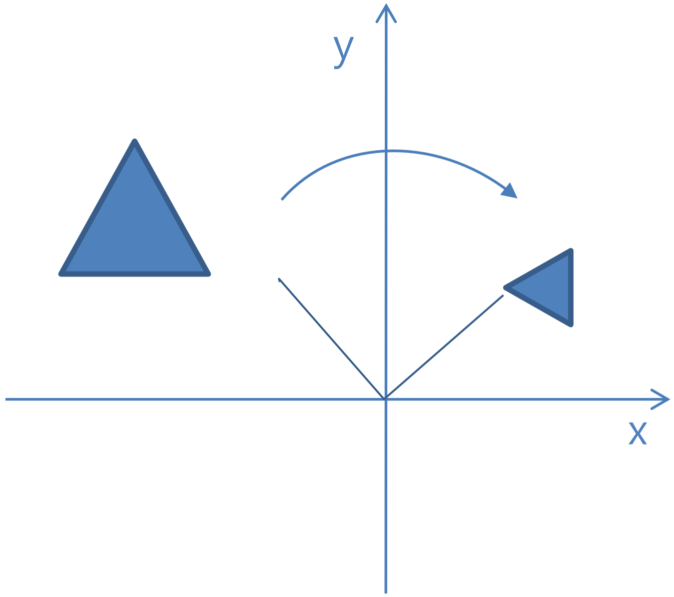
  - x' =

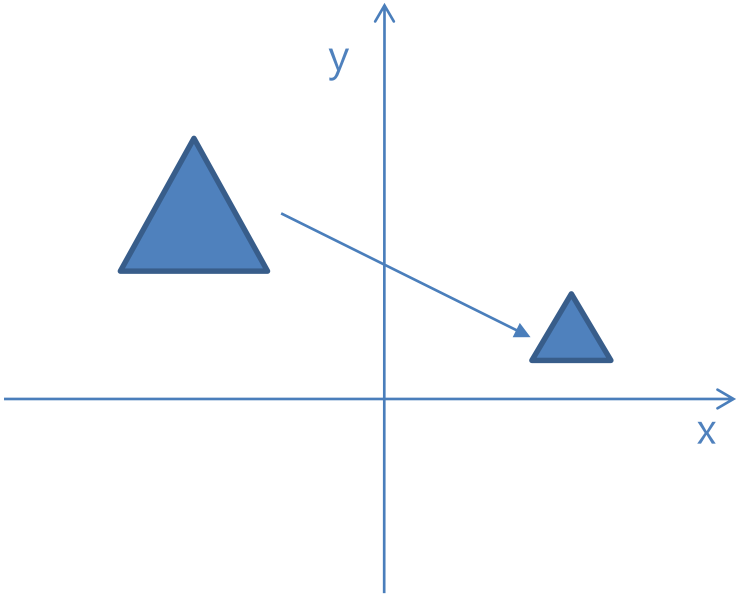
  - y' =



기본 기하 변환

- **기하 변환 예)**





- 시계반대방향으로 90도, 2배 스케일
- 삼각형 좌표값 (-10, 5) (-5, 5) (-8, 10)
  → 변환 좌표값은?

- ½ 배 스케일, X축으로 10, Y축으로 -5 이동
- 삼각형 좌표값 (-10, 5) (-5, 5) (-8, 10)
  → 변환 좌표값은?

동차 좌표계 (Homogeneous Coordinate System)

- **동차 좌표계**
  - n차원의 공간을 n+1의 좌표로 표현하는 좌표계
  - 여러 단계의 변환행렬을 하나로 결합하여 표현할 수 있게한다.
  - 순차적인 기하변환을 처리할 때 각 단계별 좌표 값을 구하지 않고 바로 계산 하려면 행렬의 합(A)을 제거해야 함
    - $P_2 = M \cdot P_1 + A_1$
    - $P_3 = M_2 P_2 + A_2 = M_2(M_1 P_1 + A_1) + A_2$

      $= M_2 M_1 P_1 + M_2 A_1 + A_2$

  - 동차 좌표계를 이용하여 기본 변환을 행렬 곱으로만 표현한다 → 변환을 간단히 처리, 계산량을 줄일 수 있다. 즉,

    $$\begin{aligned} P_n &= M_{n-1} \cdot P_{n-1} = M_{n-1} \cdot M_{n-2} \cdot P_{n-2} = \ldots \\ &= M_{n-1} \cdot M_{n-2} \cdot \ldots \cdot M_1 \cdot P_1 \\ &= M \cdot P_1 \end{aligned}$$

  - 2차원의 데이터 v = (v1, v2)가 있을 때, 이 데이터의 동차 좌표는 V′ = (v1, v2, p)
    - p = 0 -> v′는 벡터
    - p ≠ 0 -> v′는 점
    - 동차 좌표계에서 다음의 점들은 모두 같은 점이다.
      - (5, 1, 1) = (10, 2, 2) = (15, 3, 3) = (20, 4, 4)…

동차 좌표계 (Homogeneous Coordinate System)

- **행렬식**
  - 2차원의 점 P(x, y)를 동차 좌표계로 표현하면 차원이 하나 증가된다. → 즉, P(hx, hy, h) ($h \neq 0$) → h는 임의의 값 → 한 점은 동차 좌표계에서 h의 값에 따라 여러 개의 좌표로 표현될 수 있다.
    - 동차 좌표계에서 한 점의 좌표 P(X, Y, h)로 주어지면 → 2차원 기하 평면에서 $P(\frac{X}{h}, \frac{Y}{h})$로 대응된다.
    - $P(x_1, y_1, h_1)$, $P(x_2, y_2, h_2)$로 주어질 때, $(\frac{x1}{h1}, \frac{y1}{h1}) = (\frac{x2}{h2}, \frac{y2}{h2})$ 이면 2차원 기하 평면에서 동일한 점이 된다.
    - h = 1 이면 (x, y) → (x, y, 1)
  - 이동의 3차원 행렬: $T(t_x, t_y) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$
  - 회전의 3차원 행렬: $R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$
  - 신축의 3차원 행렬: $S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$

동차 좌표계 (Homogeneous Coordinate System)

- 연속된 기본 변환을 동차 좌표계를 사용하여 하나의 행렬로 나타낼 수 있다. 하나의 변환 행렬로 표현한 합성 변환에서는 한번의 행렬 곱셈만 필요하다
  - 이동: 연속적으로 2번 이동하는 경우

    $P' = T(t_{x2}, t_{y2}) \bullet T(t_{x1}, t_{y1}) \bullet P$

    $= T(t_{x2}+t_{x1}, t_{y2}+t_{y1}) \bullet P$
  - 신축: 연속적으로 2번 신축 하는 경우

    $P' = S(s_{x2}, s_{y2}) \bullet S(s_{x1}, s_{y1}) \bullet P$

    $= S(s_{x2} \cdot s_{x1}, s_{y2} \cdot s_{y1}) \bullet P$
  - 회전 : 연속적으로 2번 회전하는 경우

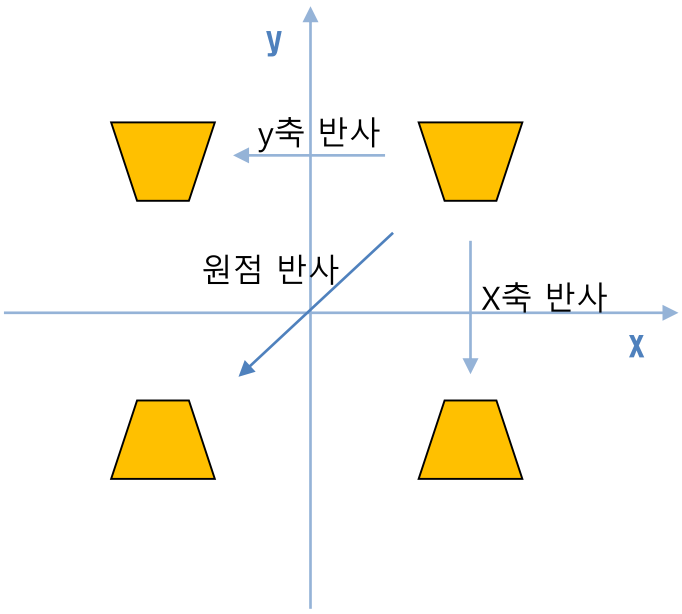
    $P' = R(\theta_2) \bullet R(\theta_1) \bullet P$

    $= R(\theta_2 + \theta_1) \bullet P$
  - 임의의 점 (x_0, y_0)에 대하여 신축 (s_x, s_y)하는 경우
    - $P' = T(x_0, y_0) \cdot S(s_x, s_y) \cdot T(-x_0, -y_0) \cdot P$

  - 행렬의 곱셈에서 결합법칙은 성립한다.
    - $(A \cdot B) \cdot C = A \cdot (B \cdot C)$
  - 교환 법칙은 성립하지 않는다.
    - $A \cdot B \neq B \cdot A$
  - 단, 같은 종류의 기하변환에서 교환 법칙은 성립한다.

기타 기하 변환: 반사

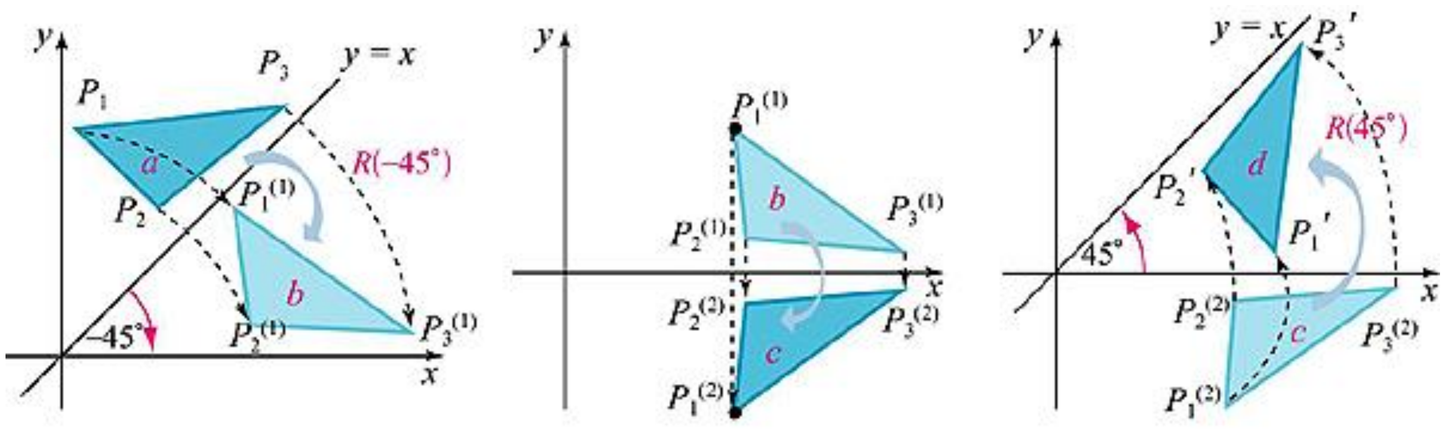
- Reflection (반사): 거울 영상
  - 고정점에 대하여 객체가 반대방향으로 변환되는 것
  - y=0 (x축)에 대하여 반사
    - y 좌표값 부호 변경
  - x=0 (y축)에 대하여 반사
    - x 좌표값 부호 변경
  - 원점 (0,0)에 대하여 반사
    - 모두 변경



기타 기하 변환: 반사

- y = x에 대한 반사
  - 시계방향으로 45' 회전
  - X축에 대하여 반사
  - 시계 반대 방향으로 45' 회전

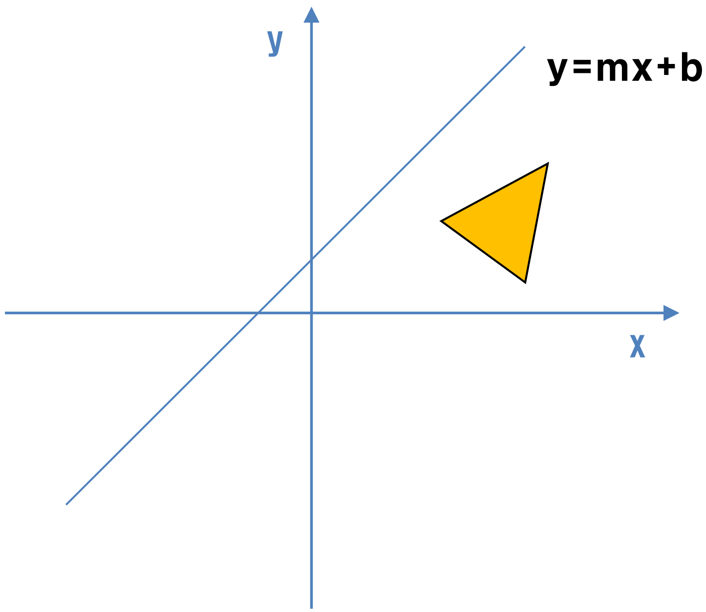
- y = -x에 대한 반사



(a) 다각형과 반사축을 회전 (b) 반사축을 기준으로 반사 (c) 반대방향으로 회전

기타 기하 변환: 반사

- y = mx + b에 대하여 반사



- 예) y = 1.732x + 2 직선에 대하여 선분 (0, 5) (2, 8) 반사 (1.732= $3/\sqrt{3}$)

기타 기하 변환: 밀림

- **밀림 (Shearing)**
  - 2차원 평면상에서 객체의 한 부분을 고정시키고 다른 부분을 밀어서 생기는 변환
    - 고정된 지점에서 멀수록 밀리는 거리가 커진다. (고정된 지점과의 거리에 비례하여 밀리는 경우가 결정된다)
    - x축에 대한 밀림:

      $x' = x + h_x \bullet y$ (h_x: 밀림 비율)

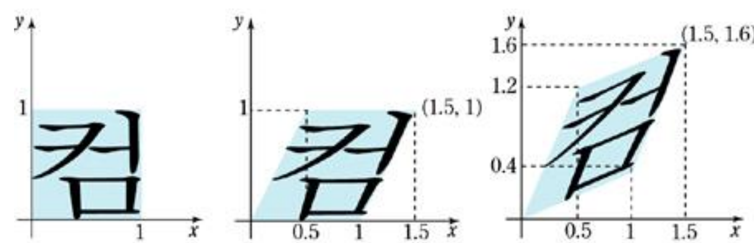
      $y' = y$
    - y축에 대한 밀림:

      $x' = x$

      $y' = y + h_y \bullet x$ (h_y :밀림 비율)

  - 행렬을 사용하면,

    P′ =



(a) 원래 글자 (b) x방향으로 0.5 밀림 (c) y방향으로 0.4 밀림

윈도우와 뷰포트

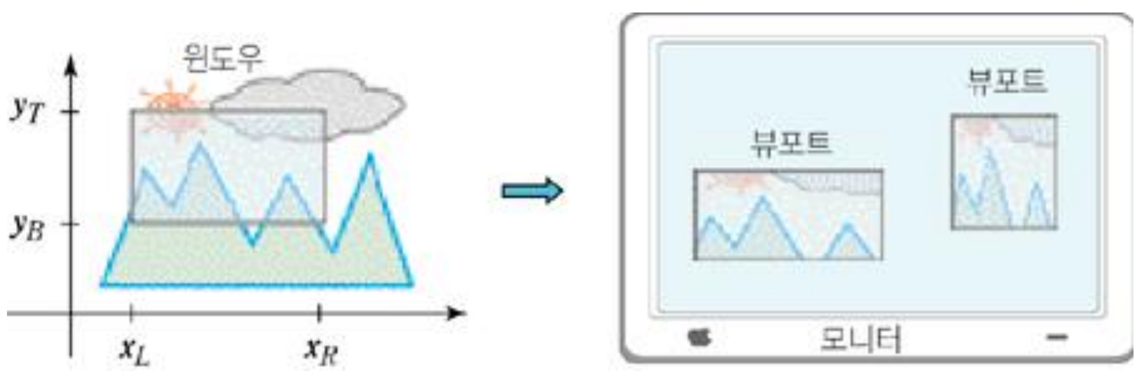
- **2차원 그래픽스의 뷰잉 파이프라인**
  - 모델 좌표계: 개별 객체를 표현하기 위해 사용되는 좌표계
  - 월드 좌표계: 각 모델 좌표계의 통합된 좌표계
  - 뷰잉 좌표계: 출력장치에 출력 위치 및 크기 설정하여 뷰포트에 출력, 뷰포트 좌표계
  - 정규 좌표계: 정규화된 좌표계
  - 장치 좌표계: 출력하려는 장치 좌표계

윈도우와 뷰포트

- Window
  - 출력 장치에 표시하기 위해 선택된 세계 좌표 영역

- Viewport
  - 윈도우가 사상되는 출력 장치의 영역

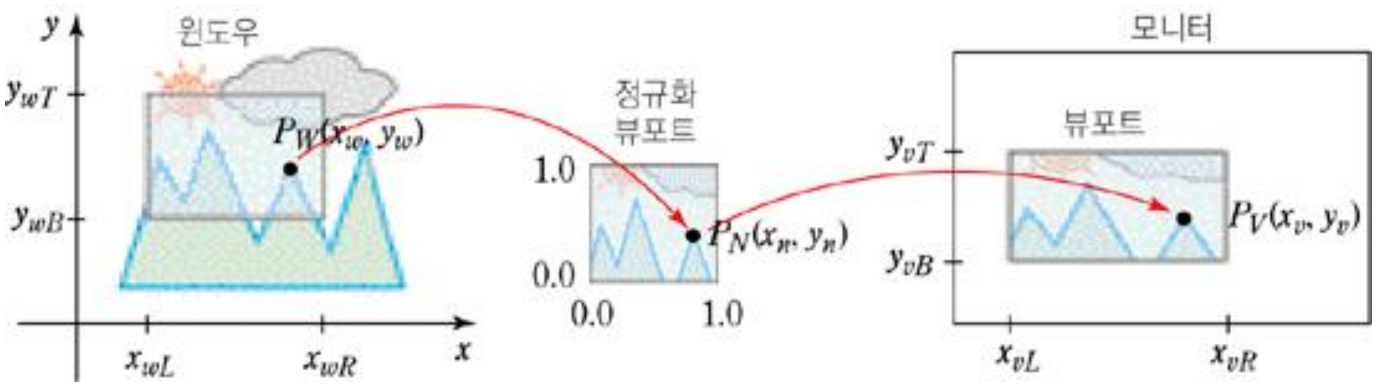
- **윈도우-뷰포트 변환에 의한 효과**
  - Zooming 효과: (zoom in/zoom out)
  - Panning 효과: 카메라 각도를 돌려가면서 비디오 촬영하는 것과 같은 효과
  - 한번에 여러 개의 화면을 가질 수 있다.



윈도우와 뷰포트

- **윈도우-뷰포트 좌표 변환**
  - (x_w, y_w): 윈도우 내의 점 (x_v, y_v): 뷰포트 안의 점
    - $x_v = x_{vL} + (x_w - x_{wL})s_x$, $s_x = \dfrac{(x_{vR} - x_{vL})}{(x_{wR} - x_{wL})}$
    - $y_v = y_{vB} + (y_w - y_{wB})s_y$, $s_y = \dfrac{(y_{vT} - y_{vB})}{(y_{wT} - y_{wB})}$

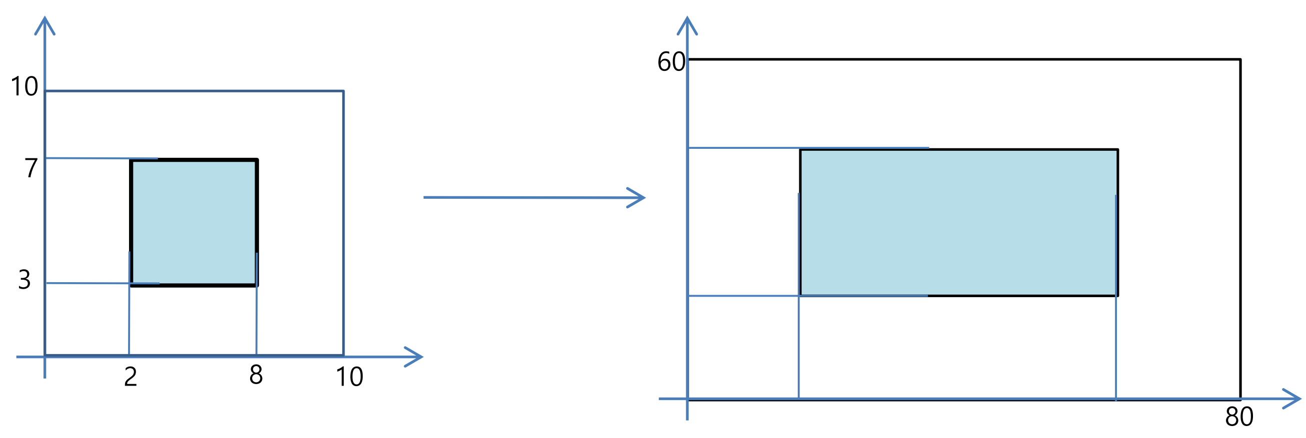
x_{wR}, x_{wL}: 윈도우의 x방향 최대값, 최소값
y_{wT}, y_{wB}: 윈도우의 y방향 최대값, 최소값



윈도우와 뷰포트

- **예) 다음의 도형에 대하여 윈도우-뷰포트 변환이 주어졌을 때 변환 좌표 값**
  - 윈도우 (0, 10, 0, 10) → 뷰포트 (0, 80, 0, 60)

    도형 좌표: (2, 3) (8, 7)로 이루어진 사각형이 윈도우-뷰포트 변환 후 좌표값:



Clipping (클리핑)

- **Clipping의 개념**
  - 윈도우-뷰포트 변환 시, 출력장치에 표시되어서는 안될 그림영역을 제거한 뒤, 나머지 그림영역을 출력화면에 나타내는 것

  - 월드 좌표 클리핑:
    - 윈도우를 설정할 때 윈도우 바깥 영역을 제거하여 윈도우 내부 영역만 뷰포트로 매핑시키는 방법
  - 뷰포트 클리핑:
    - 월드 좌표계를 표현된 그림 전부를 뷰포트로 매핑시킨 후 뷰포트 외부에 위치한 객체나 그림의 일부를 제거하는 방법

  - 두 클리핑이 모두 결과는 같다.
  - 월드 좌표계를 사용하면 계산 시간이 줄어든다.

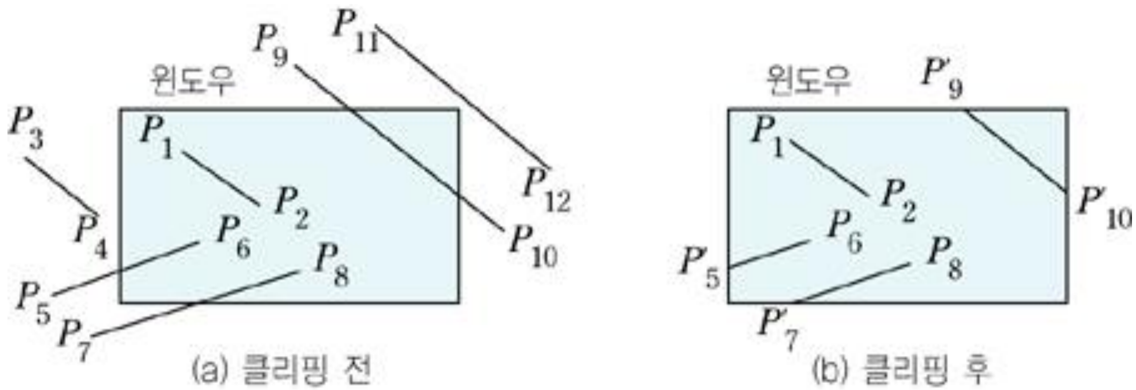
Clipping (클리핑): 점

- **점 클리핑**
  - 클리핑 되는 객체가 점
  - 한 점 P(x, y)는
    - $x_L \le x \le x_R$, $y_B \le y \le y_T$

    이면 그려진다.

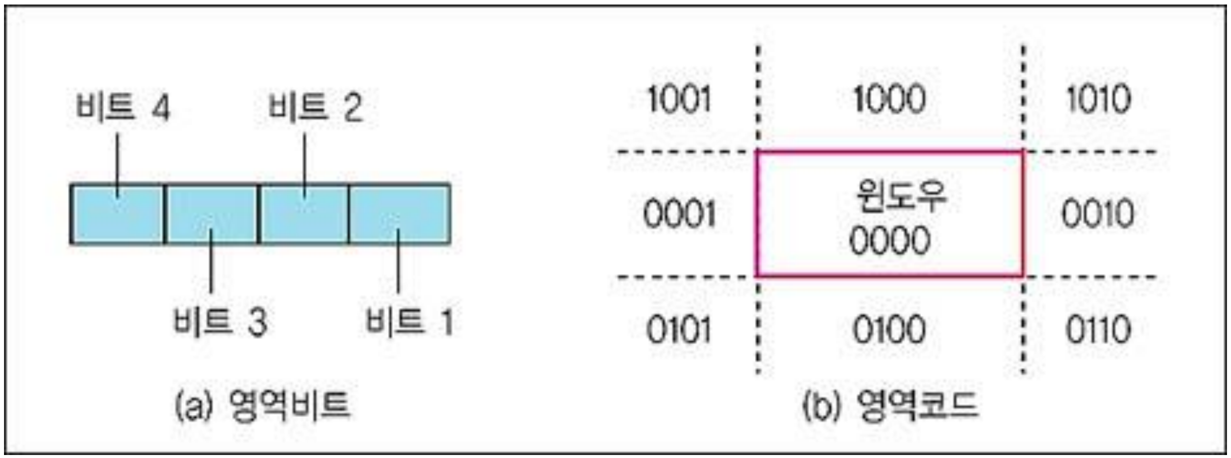
Clipping (클리핑): 선

- **선 클리핑**
  - 클리핑 되는 객체가 선분
  - 선분이 클리핑 영역의 내부 또는 외부에 완전히 포함되는가/포함되지 않는가
  - 부분적으로 속하는가
  - 속한다면 교차점은 어떻게 구하는가



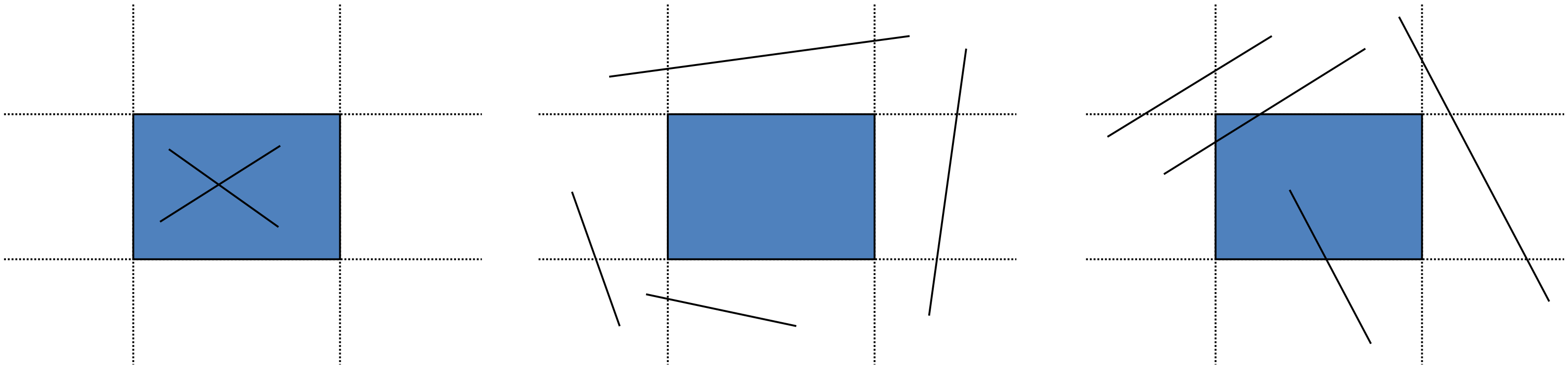
Clipping (클리핑): 선

- **Cohen-Sutherland 알고리즘**
  - 윈도우를 중심으로 전체 그림 영역을 9개 영역으로 구분
  - 각 영역에 4비트를 사용하여 영역코드를 부여한다.
    - 비트 1: 윈도우의 왼쪽에 있으면 1
    - 비트 2: 윈도우의 오른쪽에 있으면 1
    - 비트 3: 윈도우의 아래쪽에 있으면 1
    - 비트 4: 윈도우의 위쪽에 있으면 1



Clipping (클리핑): 선

- 알고리즘 수행 과정
  - 양 끝점의 코드가 모두 0000이면 →
  - 양 끝점의 코드 중 한 쪽 코드는 0이고 다른 쪽 코드는 0이 아니면 →
  - 양 끝점 코드가 모두 0이 아니고, 양 끝점 코드간 AND 연산이 0이 아니면 →
  - 양 끝점 코드가 모두 0이 아니고, 양 끝점 코드간 AND 연산이 0이면 →



Clipping (클리핑): 선

- **주어진 선분에 대한 교차점 구하기**
  - 수직 경계: $x = x_L$ 또는 $x = x_R$

    $$y = y1 + m(x - x1), \quad m = \frac{y2 - y1}{x2 - x1}$$

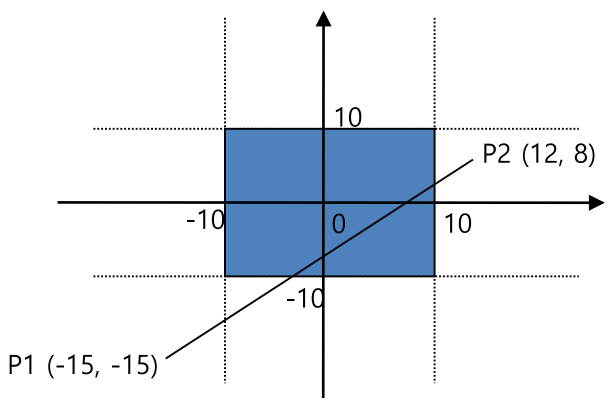
  - 수평 경계: $y = y_B$ 또는 $y = y_T$

    $$x = x1 + (y - y1)/m$$

    $$m = \frac{y2 - y1}{x2 - x1}$$

    x1, y1: 선분의 끝 점

    x_L, x_R , y_B, y_T : 윈도우의 경계



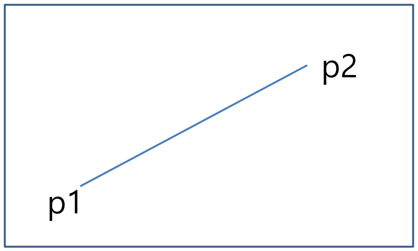
Clipping (클리핑): 선

- **Liang-Barsky 알고리즘**
  - 매개변수 방정식을 이용하여 선분을 윈도우 경계에 대하여 자르는 알고리즘
  - 선을 나타내는 매개변수 방정식은
    - p(u) = (1- u)p1 + up2

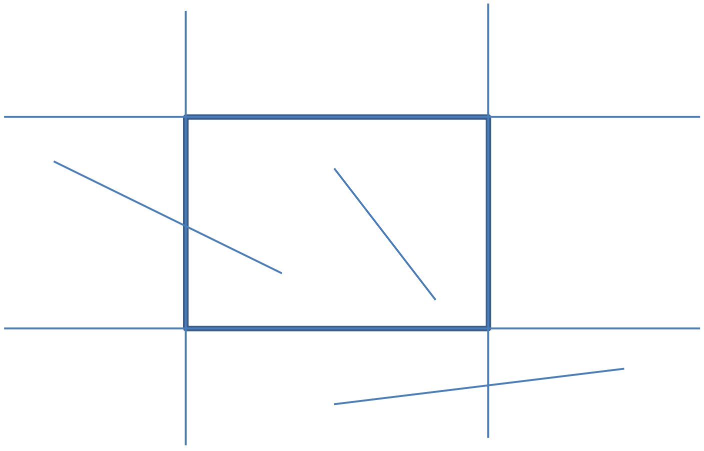
      = p1 + u(p2 - p1)   0 <= u <= 1

      즉, x = x1 + u(x2 - x1)

      y = y1 + u(y2 - y1)



    - u = 0 -> (x, y) = (x1, y1)
    - u = 1 -> (x, y) = (x2, y1)
    - 그 외의 값이면, (x1, y1)과 (x2, y2)를 벗어난 연장 선상의 한 점의 값



Clipping (클리핑): 선

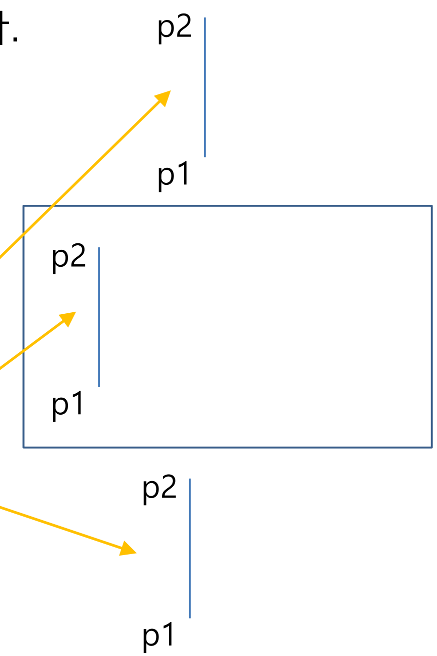
- **끝점이 P1 = (x1, y1) P2 = (x2, y2) 인 선분일 때**
  - 매개변수 방정식 사용하여 임의의 점 P (x, y) 을 표시
    - $x = x1 + (x2 - x1)u$ $\quad 0 \le u \le 1$ $\quad$ (x2 - x1 -> d_x)
    - $y = y1 + (y2 - y1)u$ $\quad 0 \le u \le 1$ $\quad$ (y2 - y1 -> d_y)
      - u = 0 $\rightarrow$ p = p1 $\rightarrow$
      - u = 1 $\rightarrow$ p = p2 $\rightarrow$
    - 선분위에 있는 모든 점들은 아래의 조건을 만족
      - $x_{min} \le x \le x_{max}$, $\quad y_{min} \le y \le y_{max}$
    - 매개 변수 방정식으로 다시 작성하면
      - $x_{min} \le x1 + d_x u \le x_{max}$ --- ① $\quad (d_x = x2 - x1)$
      - $y_{min} \le y1 + d_y u \le y_{max}$ --- ② $\quad (d_y = y2 - y1)$
      - ① 은 다음과 같다
        - » 왼쪽 가장자리에 대하여 $\quad -d_x u < x1 - x_{min}$
        - » 오른쪽 가장자리에 대하여 $\quad d_x u < x_{max} - x1$
      - ② 도 같은 방식으로 바꿀 수 있다.
        - » 아래쪽 가장자리에 대하여 $\quad -d_y u < y1 - y_{min}$
        - » 위쪽 가장자리에 대하여 $\quad d_y u < y_{max} - y1$

Clipping (클리핑): 선

- 위의 식은, $up_k < q_i$, $k = 1, 2, 3, 4$
  - $p_1 = -d_x$ $q_1 = x1 - x_{min}$ left
  - $p_2 = d_x$ $q_2 = x_{max} - x1$ right
  - $p_3 = -d_y$ $q_3 = y1 - y_{min}$ bottom
  - $p_4 = d_y$ $q_4 = y_{max} - y1$ top

- **클리핑 조건**
  - $p_k == 0$ 이면, 선분은 네 경계선 중 하나에 평행
    - if ($p_1 == 0$) && ($q_1 = x1 - x_{min}$) < 0 → $x1 < x_{min}$ → 영역 밖에 있다.
    - if ($p_2 == 0$) && ($q_2 = x_{max} - x1$) < 0 → $x_{max} < x1$ →
    - if ($p_3 == 0$) && ($q_3 = y1 - y_{min}$) < 0 → $y1 < y_{min}$ →
    - if ($p_4 == 0$) && ($q_4 = y_{max} - y1$) < 0 → $y_{max} < y1$ →

    - if ($p_i == 0$) && ($q_i < 0$) → reject
    - if ($p_i == 0$) && ($q_i > 0$) → 한 영역은 내부에 있다
      - 다른쪽 영역에서 max (x1, x2) < x_{min} -> reject
      - 다른쪽 영역에서 min (x1, x2) < x_{max} -> reject
      - x_{min} 또는 x_{max}를 만나므로 두 점을 사용하여 클리핑



Clipping (클리핑): 선

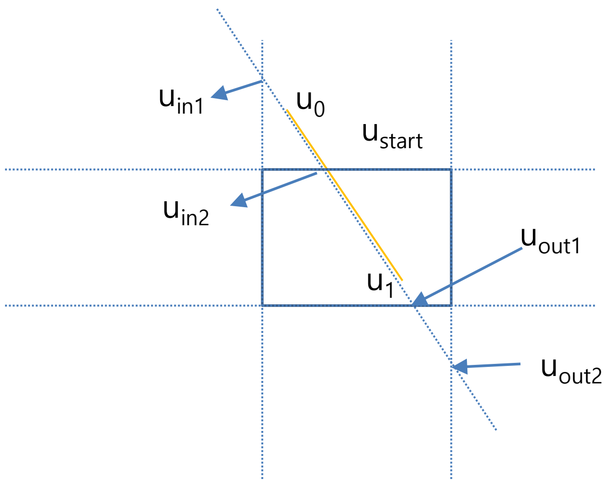
- **클리핑 조건**
  - p_k != 0 이면, 선분이 경계선 중 하나와 평행하지 않다. -> 그 선분의 무한한 연장선은 윈도우의 네 개의 경계선과 어디에선가 교차한다.
    - $p_k < 0$ -> 선분의 연장선은 해당 경계선을 밖 -> 안으로 뚫고 들어온다.
    - $p_k > 0$ -> 선분의 연장선은 해당 경계선을 안 -> 밖으로 뚫고 나간다.

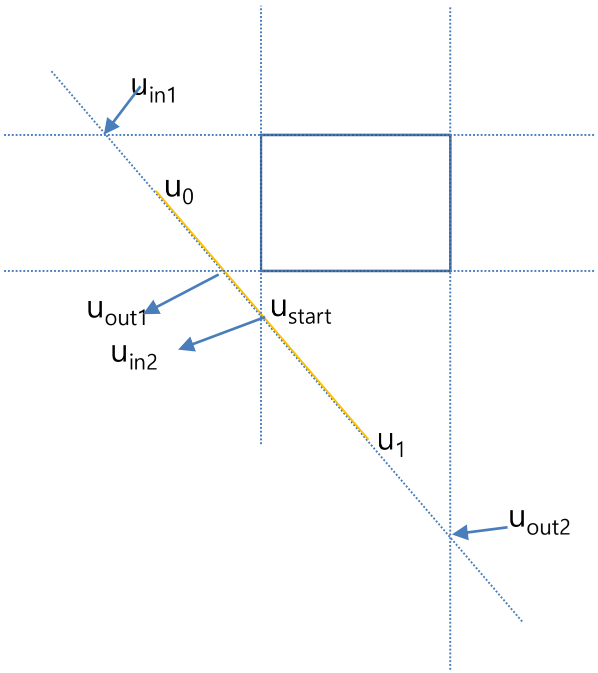
    - $u_k = q_k / p_k$ 에 의해서 뚫고 들어오는 두 개의 u값과 0 중에서 가장 큰 값 -> ustart
    - 경계선을 뚫고 나가는 두 개의 u 값과 1 중에서 가장 작은 값 -> uend
      - if $u_{start} > u_{end}$ -> reject
      - it $u_{start} < u_{end}$ -> 새로운 좌표값

        new_x1 = x1 + u_{start} * dx; new_y1 = y1 + u_{start} * dy;

        new_x1 = x2 + u_{end} * dx; new_y2 = y1 + u_{end} * dy;

Clipping (클리핑): 선



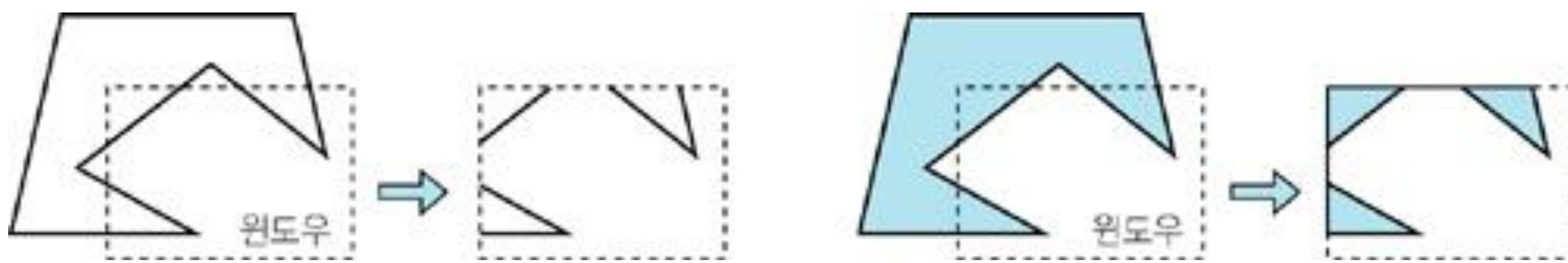


- **이 알고리즘의 장점**
  - 필요할 때까지 교차점의 계산을 피할 수 있다.
  - 코헨 서덜랜드 알고리즘에 비해
    - 선분 줄임을 여러 번 반복 -> 클리핑 알고리즘의 재실행을 피할 수 있다.

Clipping (클리핑): 다각형

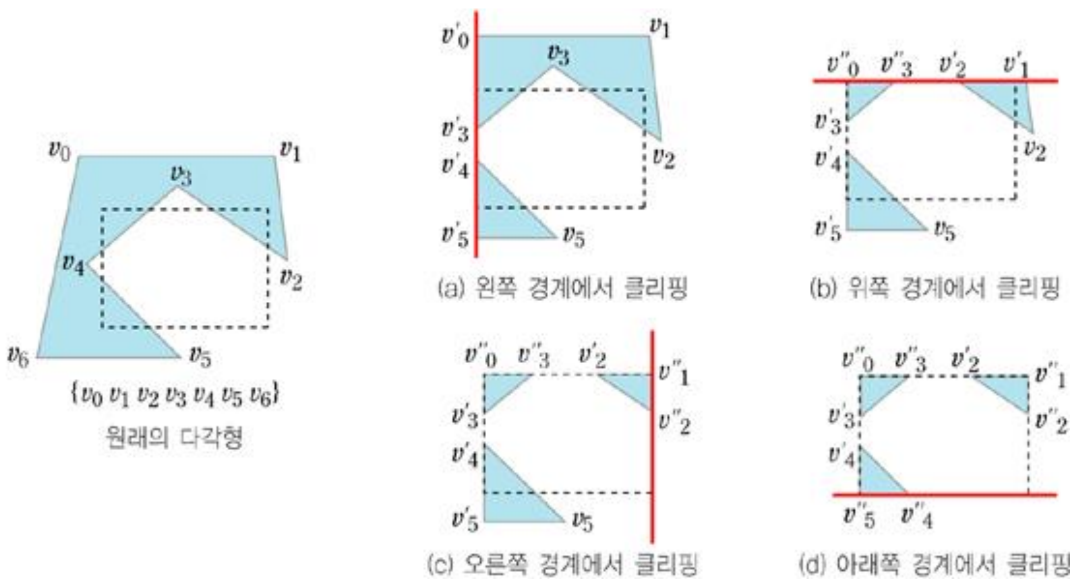
- **속이 빈 다각형(Hollow polygon) :**
  - 선 클리핑 알고리즘 적용

- **속이 찬 다각형 :**
  - 몇 개의 Closed filled polygon 생성



Clipping (클리핑): 다각형

- Sutherland-Hodgeman 알고리즘
  - 다각형의 모든 꼭지점이 윈도우의 내부 또는 외부에 완전히 포함되는지를 결정하여 다각형 전체를 제거하거나 선택하고 그 외의 경우에는 다음 알고리즘을 적용하여 다각형을 클리핑
  - 한 경계변을 기준하여 이 변이 윈도 바깥쪽 영역에 속하는 다각형 부분은 클리핑 소거



Clipping (클리핑): 다각형

- 각 윈도우 경계 (상하좌우)에 대하여 다음 알고리즘을 적용
- 4가지 경우로 구분하여 다각형 꼭지점을 재구성