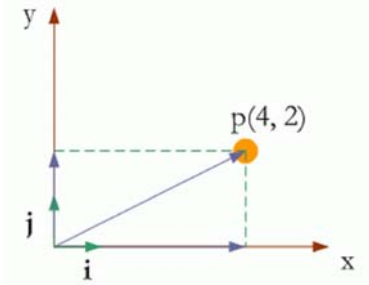
8장. 가시성 판단

- 학습목표
  - 후면제거의 정의와 처리방법을 이해한다.
  - 절단작업의 정의와 처리방법을 이해한다.
  - 지엘의 절단 방법을 이해한다.
  - 은면제거의 정의를 이해한다.
  - 지-버퍼 알고리즘을 구체적으로 이해한다.

1

8.1 벡터

- 정규화 벡터(Normalized Vector)
  - 벡터의 크기(절대값)



[그림 8-1] 좌표벡터

$$| \boldsymbol{p} | = \sqrt{x^2 + y^2 + z^2}$$

$$\boldsymbol{p}' = (\frac{x}{| \boldsymbol{p} |}, \frac{y}{| \boldsymbol{p} |}, \frac{z}{| \boldsymbol{p} |})$$

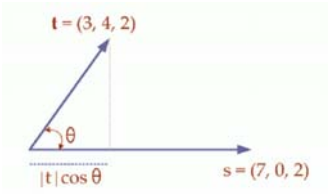
$$= (\frac{x}{\sqrt{x^2 + y^2 + z^2}}, \frac{y}{\sqrt{x^2 + y^2 + z^2}}, \frac{z}{\sqrt{x^2 + y^2 + z^2}})$$

2

벡터 내적과 외적

- 내적(**Inner Product, Dot Product**)

$$s \cdot t = |s||t|\cos\theta = s_x t_x + s_y t_y + s_z t_z$$



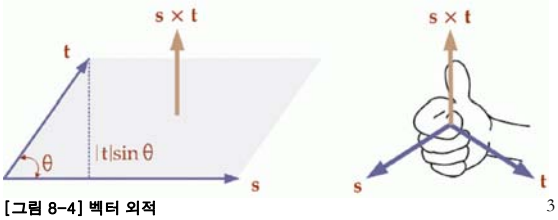
[그림 8-3] 벡터 내적

- 외적(**Outer Product, Cross Product**)

$$s \times t = |s||t|\sin\theta\, n$$
$$= (s_y t_z - s_z t_y,\ -s_x t_z + s_z t_x,\ s_x t_y - s_y t_x)$$

$$s \times t = -t \times s$$

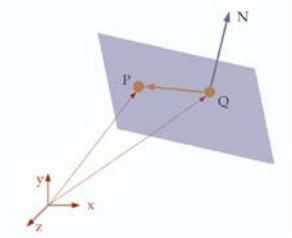
- 정규화 법선벡터
  =정규화 외적벡터



[그림 8-4] 벡터 외적

평면 표현

$$(P - Q) \cdot N = 0$$
$$P \cdot N = Q \cdot N$$



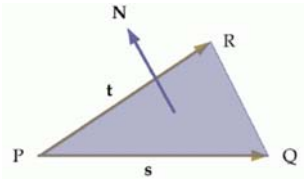
[그림 8-5] 평면 표현

$$(x, y, z) \cdot (A, B, C) = Q \cdot N$$
$$Ax + By + Cz = Q \cdot N$$
$$Ax + By + Cz + D = 0$$

의미는?

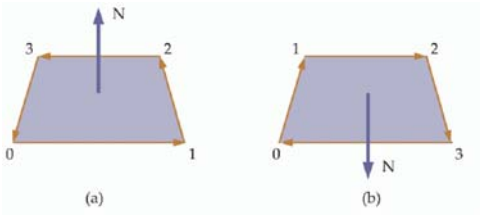
지엘의 법선벡터

$$s = (Q_x - P_x, Q_y - P_y, Q_z - P_z)$$
$$t = (R_x - P_x, R_y - P_y, R_z - P_z)$$
$$N = s \times t \qquad (8.11)$$



[그림 8-7] 법선벡터

- 법선벡터 방향
  - 오른 손을 명시된 정점 순으로 감싸 쥐었을 때 엄지방향

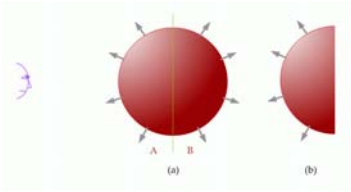


[그림 8-8] 법선벡터 방향

5

8.2 후면제거–후면

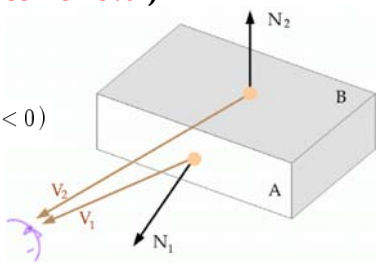
- 전면과 후면
  - 후면(Back-Facing Polygon)
  - 전면(Front-Facing Polygon)



[그림 8-9] 전면과 후면

- 후면제거(Backface Culling, Backface Removal)
  - 시점과 면의 오리엔테이션만으로 판단
  - 보이지 않는 면의 거의 절반을 제거

$$Backface = (N \cdot V < 0) = (|N||V|\cos\theta < 0)$$

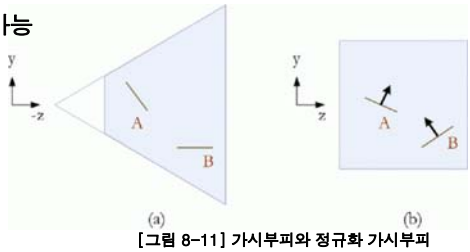


[그림 8-10] 시점벡터와 법선벡터

6

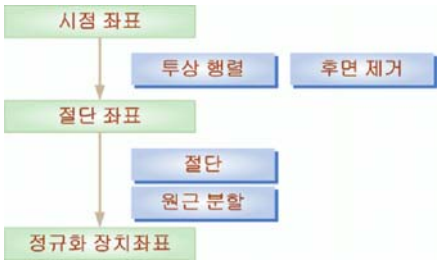
지엘의 후면제거

- 정규화 가시부피
  - 법선벡터의 z 값만으로 판단가능



[그림 8-11] 가시부피와 정규화 가시부피

- glEnable(GL_CULL_FACE);
- glCullFace(GL_FRONT);
- glDisable(GL_CULL_FACE);



[그림 8-12] 지엘의 후면제거

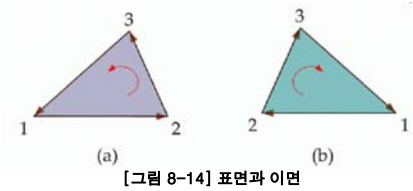
7

표면과 이면

- 하나의 면 = 표면 + 이면

[그림 8-13] 표면과 이면

- 표면
  - 시계방향으로 정의된 면
    - glFrontFace(GL_CCW)
  - 반시계방향으로 정의된 면
    - glFrontFace(GL_CW)



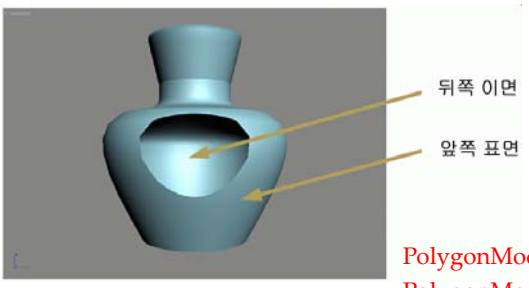
[그림 8-14] 표면과 이면

- pp. 369 그림 참조

8

표면과 이면

- 후면의 이면
  - 시점이 결정되면 다각형의 표면과 이면 중 하나의 면만 보임.
  - 지엘은 표면과 이면 중 하나만을 선택하여 그 면으로 해당 다각형을 대신 함
- 후면이면 = 표면



[그림 8-16] 후면의 이면

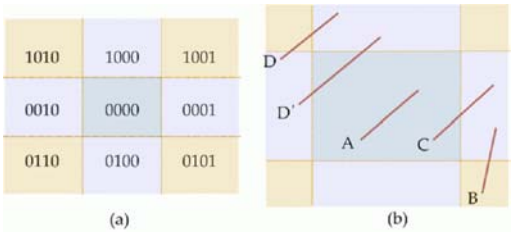
PolygonMode(GL_FRONT, GL_FILL)
PolygonMode(GL_BACK, GL_LINE)
PolygonMode(GL_BACK, GL_POINT)
// GL_FRONT_AND_BACK

8.3 절단 알고리즘–절단(Clipping)

- 2차원 절단
  - 윈도우(Window), 뷰포트(Viewport), 시저 박스(Scissor Box)
- 3차원 절단
  - 가시부피(View Volume)
- 절단 다각형
  - 절단 사각형(Clip Rectangle)

코헨-서더런드 알고리즘

- 4비트 아웃코드(Outcode)



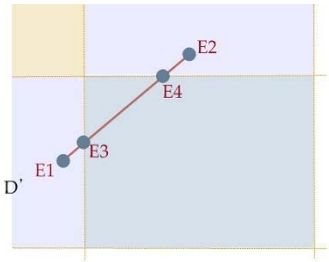
[그림 8-18] 코헨-서더런드 알고리즘

- 테스트 1) E1 = E2 = 0000
  - 완전히 사각형 내부 선분이므로 보이는 선분으로 판정한다. (선분 A)
- 테스트 2) E1 & E2 != 0000
  - 선분이 온전히 절단 사각형 밖에 있으므로 제거한다. (선분 B)
- 테스트 3) E1 != 0000, E2 = 0000 (또는 그 반대)
  - 교차점 계산에 의해 절단한다. (선분 C)
- 테스트 4) E1 & E2 = 0000
  - 양끝점이 모두 절단 사각형 밖에 있지만 서로 다른 선분이다.
  - 교차점 계산에 의해 절단한다. (선분 D, D')

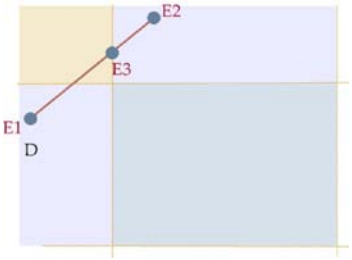
11

코헨-서더런드 알고리즘

- 선분분할



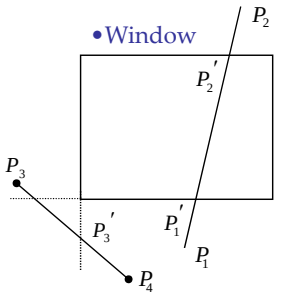
[그림 8-19] 선분분할 I

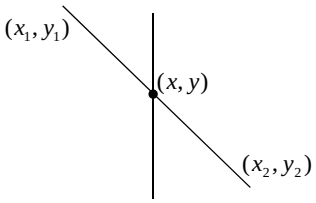


[그림 8-20] 선분분할 II

- 분할된 선분을 대상으로 다시 테스트
  - 선분 D: E3 & E2 != 0000 이므로 온전히 외부 선분으로 무시
- 차원을 확장하는 방법은?

12





- (Code1 & Code2) == 0000 : accept
- 1 at the same bits : reject the line
  - *ex) 1001 - 0101*

- 교차점: Line Equation이용

$$y = y_1 + m(x - x_1)$$

- 수직 경계와의 교차

$$x = x_1 + \frac{y - y_1}{m}$$

- 수평 경계와의 교차

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

13

- Line clipping algorithm
  - encode
  - swap
  - accept
  - reject
  - successive finding crossing points (/, *)
  - and update line endpoints

14

```
Var
      xw_min, xw_max, yw_min, yw_max: real;

Procedure clipCohSuther (x1, y1, x2, y2: real)
      type
        boundaries = (left, right, bottom, top);
        code = array [boundaries] of boolean;
      var
        code1, code2: code;
        done, display: boolean;
        m: real;

      Procedure encode (x, y: real, var c: code)
        begin
              if x < xw_min then c[left] = true;
              else c[left] = false;
              if x < xw_max then c[right] = true;
              else c[left] = false;
              if y < yw_min then c[bottom] = true;
              else c[left] = false;
              if y < yw_max then c[top] = true;
              else c[left] = false;
        end {encode}
      Procedure swap_if_nedded (x1,y1, x2, y2,  c1, c2)
              // x1, y1이 외부에 있도록 바꿈

      function accept (c1, c2) : boolean;
        var k: boundaries;
        begin
              accept = true;
              for k=left to top do
                if c1[k] or c2[k] then accept = false;
        end {accept}
```

```
function reject (c1, c2) : boolean;
  var k: boundaries;
  begin
        reject = false;
        for k=left to top do
          if c1[k] and c2[k] then reject = true;
  end {accept}

begin {clipALine}
    done = false;
    display = false;
    while not done do begin
        encode (x1, y1, code1);
        encode (x2, y2, code2);
        if accept(code1, code2) { done=true; display=true; }
        else if reject (code1, code2)
                  { done=false; display=false; }
        else {
          swap_if_needed (x1, y1, x2, y2, code1, code2);
          m = (y2-y1) / (x2-x1);
          if code1[left] then
            { y1 = y1 + (xw_min-x1) * m; x1 = xw_min; }
          else if code1[right] then
            { y1 = y1 + (xw_max-x1) * m; x1 = xw_max; }
          else if code1[bottom] then
            { x1 = x1 + (yw_min-y1) / m; y1 = yw_min; }
          else if code1[right] then
            { x1 = x1 + (yw_max-y1) / m; y1 = yw_max; }
        }
  end {while not done}
end {clipALine}
```

15

- Cohen-Sutherland Line Clipping Algorithm

```
/* clipCohSuth, Chapter 6, pp. 228-230 */

#include "graphics.h"

/* EXAMPLE STARTS HERE */
#define ROUND(a)    ((int)(a+0.5))

/* Bit masks encode a point's position relative to the clip edges.  A
   point's status is encoded by OR'ing together appropriate bit masks.
*/
#define LEFT_EDGE   0x1
#define RIGHT_EDGE  0x2
#define BOTTOM_EDGE 0x4
#define TOP_EDGE    0x8

/* Points encoded as 0000 are completely Inside the clip rectangle;
   all others are outside at least one edge.  If OR'ing two codes is
   FALSE (no bits are set in either code), the line can be Accepted.  If
   the AND operation between two codes is TRUE, the line defined by those
   endpoints is completely outside the clip region and can be Rejected.
*/
#define INSIDE(a)   (!a)
#define REJECT(a,b) (a&b)
#define ACCEPT(a,b) (!(a|b))
```

```
unsigned char encode (wcPt2 pt,
                      dcPt winMin, dcPt winMax)
{
  unsigned char code=0x00;

  if (pt.x < winMin.x)
    code = code | LEFT_EDGE;
  if (pt.x > winMax.x)
    code = code | RIGHT_EDGE;
  if (pt.y < winMin.y)
    code = code | BOTTOM_EDGE;
  if (pt.y > winMax.y)
    code = code | TOP_EDGE;
  return (code);
}
void swapPts (wcPt2 * p1, wcPt2 * p2)
{
  wcPt2 tmp;
  tmp = *p1; *p1 = *p2; *p2 = tmp;
}
void swapCodes (unsigned char * c1,
                unsigned char * c2)
{
  unsigned char tmp;
  tmp = *c1; *c1 = *c2; *c2 = tmp;
}
```

```
void clipLine (dcPt winMin, dcPt winMax, wcPt2 p1, wcPt2 p2)
{
  unsigned char code1, code2;
  int done = FALSE, draw = FALSE;
  float m;

  while (!done) {
    code1 = encode (p1, winMin, winMax);
    code2 = encode (p2, winMin, winMax);
    if (ACCEPT (code1, code2)) {
      done = TRUE;
      draw = TRUE;
    }
    else if (REJECT (code1, code2))
                  done = TRUE;
    else {
       /* Ensure that p1 is outside window */
       if (INSIDE (code1)) {
         swapPts (&p1, &p2);
         swapCodes (&code1, &code2);
       }
```

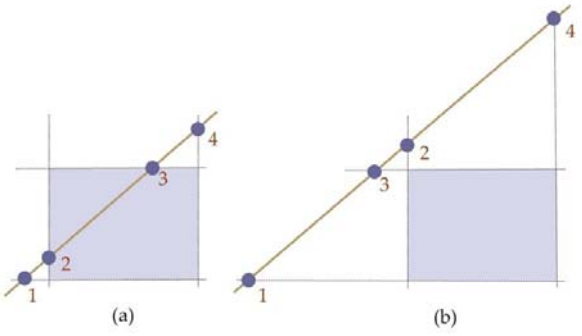
```
       /* Use slope (m) to find line-clipEdge intersections */
       if (p2.x != p1.x) m = (p2.y - p1.y) / (p2.x - p1.x);
       if (code1 & LEFT_EDGE) {
         p1.y += (winMin.x - p1.x) * m;
         p1.x = winMin.x;
       }
       else if (code1 & RIGHT_EDGE) {
          p1.y += (winMax.x - p1.x) * m;
          p1.x = winMax.x;
        }
        else if (code1 & BOTTOM_EDGE) {
           /* Need to update p1.x for non-vertical lines only */
           if (p2.x != p1.x)
               p1.x += (winMin.y - p1.y) / m;
           p1.y = winMin.y;
         }
         else if (code1 & TOP_EDGE) {
              if (p2.x != p1.x) p1.x += (winMax.y - p1.y) / m;
              p1.y = winMax.y;
            }
      }
   }
   if (draw)
     lineDDA (ROUND(p1.x), ROUND(p1.y),
     ROUND(p2.x), ROUND(p2.y));
}
```

```
#define N_PTS 5

void main (int argc, char ** argv)
{
  wcPt2 pts[N_PTS] =
    { 60, 20, 375, 80, 280, 280, 100, 280, 100, 100 };
  dcPt winMin = { 50, 50 };
  dcPt winMax = { 350, 250 };
  wcPt2 winPts[5];
  int i;
  long windowID = openGraphics (*argv, 400, 300);

  setBackground (WHITE);

  /* For illustration, draw the clipping rectangle */
  winPts[0].x = winMin.x; winPts[0].y = winMin.y;
  winPts[1].x = winMax.x; winPts[1].y = winMin.y;
  winPts[2].x = winMax.x; winPts[2].y = winMax.y;
  winPts[3].x = winMin.x; winPts[3].y = winMax.y;
  winPts[4].x = winMin.x; winPts[4].y = winMin.y;
  setColor (BLUE);
  pPolyline (5, winPts);
```

```
  /* For illustration, draw the complete set of lines */
  setColor (GREEN);
  pPolyline (N_PTS, pts);

  /* Clip pairs of points and draw line segments */
  setColor (BLACK);
  for (i=0; i<N_PTS-1; i++)
    clipLine (winMin, winMax, pts[i], pts[i+1]);

  sleep (10);
  closeGraphics (windowID);
}
```

리앙-바스키 알고리즘

- 교차점에서의 파라미터 값의 순서를 기준으로 여러 가지 경우를 판단



$0 < t_1 < t_2 < t_3 < t_4$ $0 < t_1 < t_3 < t_2 < t_4$

[그림 8-22] 리앙-바스키 알고리즘

19

Liang & Barsky's Algorithm

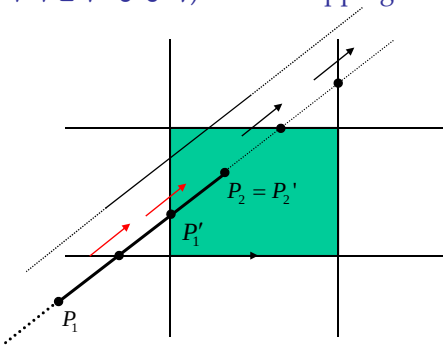
An analysis and algorithm for polygon clipping
Computer Graphics, 3, 1, (1984), 23-51
Basic Idea

- Use parametric equations(**매개변수 방정식**) for line clipping

$$\hat{P} = P_1 + u(P_2 - P_1), \quad 0 \le u \le 1$$

$$P_1' = \max_u\{P_1, \uparrow\} = \boxed{u_1}$$

$$P_2' = \min_u\{P_2, \uparrow\} = \boxed{u_2}$$

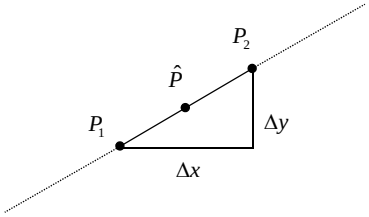


- 외부에서 내부로 들어가는 최대치
- 내부에서 외부로 나가는 최소치

20

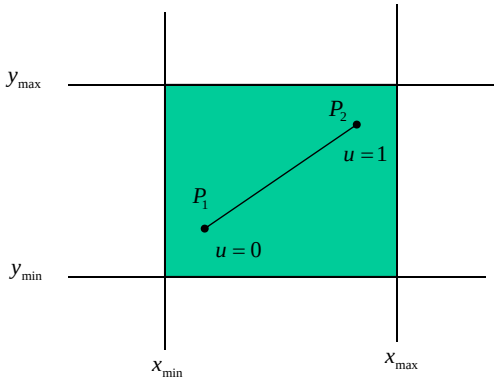
• Parametric Equation for 2D Line

$$\hat{P} = vP_1 + uP_2$$
$$u + v = 1$$
$$\therefore \hat{P} = vP_1 + uP_2$$
$$= (1-u)P_1 + uP_2$$
$$= P_1 + u(P_2 - P_1)$$



$$\hat{P} = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 + u(x_2 - x_1) \\ y_1 + u(y_2 - y_1) \end{pmatrix}$$
$$\therefore x = x_1 + u\Delta x$$
$$y = y_1 + u\Delta y$$

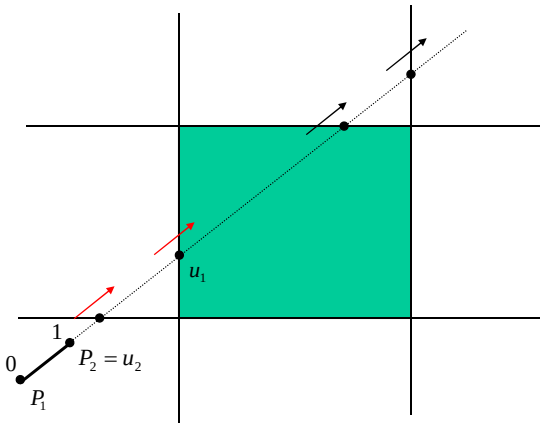
21



$$x_{min} \le x_1 + u\Delta x \le x_{max}$$
$$y_{min} \le y_1 + u\Delta y \le y_{max}$$
$$\forall 0 \le u \le 1$$

$\overline{P_1P_2}$ is contained in the window !!

22



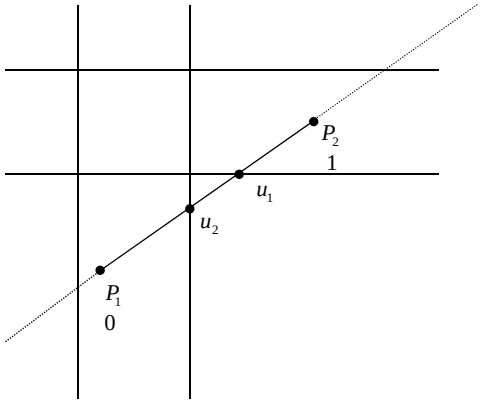
$u_1 > 1$ or $u_2 < 0$

reject $\overrightarrow{P_1P_2}$

$x = P_1 + u\Delta P$

- U1: 밖에서 안으로 들어오는 직선에 의해 결정
- U2: 안에서 밖으로 나가는 직선에 의해 결정

23



$u_1 > u_2$

24

$$\hat{P} = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 + u(x_2 - x_1) \\ y_1 + u(y_2 - y_1) \end{pmatrix}$$

$\therefore x = x_1 + u\Delta x$

$y = y_1 + u\Delta y$

$x_{\min} \le x_1 + u\Delta x \le x_{\max}$

$y_{\min} \le y_1 + u\Delta y \le y_{\max}$

$\downarrow$

$u(-\Delta x) \le x_1 - x_{\min}$

$u\overline{P}_1 \le \overline{q}_1$

$u\overline{P}_k \le \overline{q}_k, \quad k = 1,2,3,4$

$\overline{P}_1 = -\Delta x, \overline{q}_1 = x_1 - x_{\min}$

$\overline{P}_2 = \Delta x, \overline{q}_2 = x_{\max} - x_1$

$\overline{P}_3 = -\Delta y, \overline{q}_3 = y_1 - y_{\min}$

$\overline{P}_4 = \Delta y, \overline{q}_4 = y_{\max} - y_1$

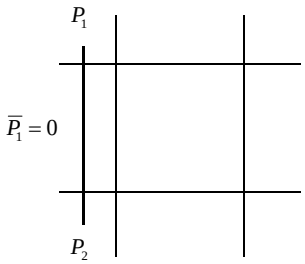
$\overline{P}_k = 0$

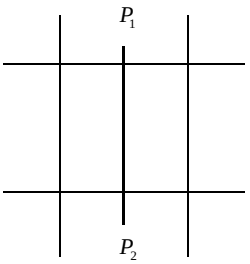
$\overline{P}_k < 0$ 3 cases

$\overline{P}_k > 0$

25

$\boxed{\overline{P}_k = 0}$ • 평행선





↑ ↑

$\overline{q}_1 < 0$

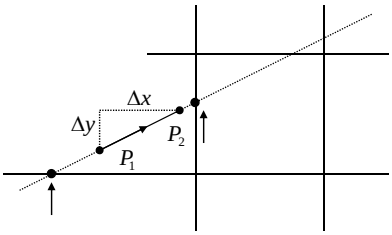
reject $\overrightarrow{P_1P_2}$

$\overline{q}_1 \ge 0$

need futher processing

26

$\overline{P}_k < 0$ •외부에서 내부로 들어감



$u\overline{P}_k \le \overline{q}_k$

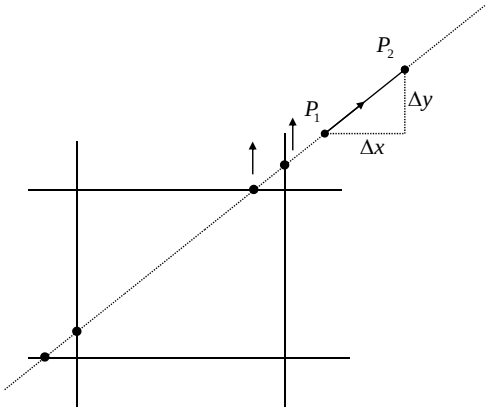
$u_k = \overline{q}_k / \overline{P}_k$

$\hat{u}_1 = \max\{0, u_k\}$

reject if $\hat{u}_1 > 1$!!!

27

$\overline{P}_k > 0$ •내부에서 외부로 나감



$u_k = \overline{q}_k / \overline{P}_k$

$\hat{u}_2 = \min\{1, u_k\}$

reject if $\hat{u}_2 < 0$!!!

28

if $\hat{u}_1 > \hat{u}_2$, rejected.

$$x_1 + \hat{u}_1 \Delta x \le x \le x_1 + \hat{u}_2 \Delta x$$
$$y_1 + \hat{u}_1 \Delta y \le y \le y_1 + \hat{u}_2 \Delta y$$

if not rejected !!!

- Algorithm
- 한쪽 면에 대해 u1, u2를 계산하여 reject조건이면 stop
- p<0이면 u1을 update하고 reject조건을 check
- p>0이면 u2를 update하고 reject조건을 check
- p=0이면 update없이 reject조건만 check

29

```
Var
    xw_min, xw_max, yw_min, yw_max: real;

Procedure clipLiangBarsky (x1, y1, x2, y2: real)
    var
     u1, u2, dx, dy: real;

    function clipTest (p, q: real, var u1, u2: real) : boolean;
     var
     r: real;
     result: boolean;
     begin {clipTest}
            result = true;
            if (p<0) {      // 외부에서 내부로 들어감
                r = q / p;
                if (r > u2) result = false;
                else if (r > u1) u1 = r;
            }
            else if {       // 내부에서 외부로 나감
                r = q / p;
                if (r < u1) result = false;
                else if (r < u2) u2 = r;
            }
            else {          // 수직 또는 수평선 (평행선)
               if (q<0) result = false;
            }
            clipTest = result;
    end {clipTest}
```

```
begin {clipLiangBarsky}
   u1 = 0;
   u2 = 1;
   dx = x2 - x1;
   dy = y2 - y1;

   if (clipTest( -dx, x1-xwmin, u1, u2) {          // left
      if (clipTest(dx, xwmax-x1, u1, u2) {         // right
         if (clipTest( -dy, y1-ywmin, u1, u2) {    // lower
            if (clipTest(dy, ywmax-y1, u1, u2) {   // upper
                        if (u2 < 1) {
                           x2 = x1 + u2 * dx;
                           y2 = y1 + u2 * dy;
                        }
                        if (u1 > 0) {
                           x1 = x1 + u1 * dx;
                           y1 = y1 + u1 * dy;
                        }
            }
         }
      }
   }
end {clipLiangBarsky}
```

- Liang & Barsky's Algorithm

```
/* clipLiangBarsky, Chapter 6, pp. 231-232 */
/* EXAMPLE STARTS HERE */
#include "graphics.h"

#define ROUND(a) ((int)(a+0.5))

int clipTest (float p, float q, float * u1, float * u2)
{
  float r;
  int retVal = TRUE;
  if (p < 0.0) {
    r = q / p;
    if (r > *u2) retVal = FALSE;
    else if (r > *u1) *u1 = r;
  }
  else if (p > 0.0) {
     r = q / p;
     if (r < *u1) retVal = FALSE;
     else if (r < *u2) *u2 = r;
  }
 /* p = 0, so line is parallel to this clipping edge */
  else if (q < 0.0) /* Line is outside clipping edge */
        retVal = FALSE;

  return (retVal);
}
```

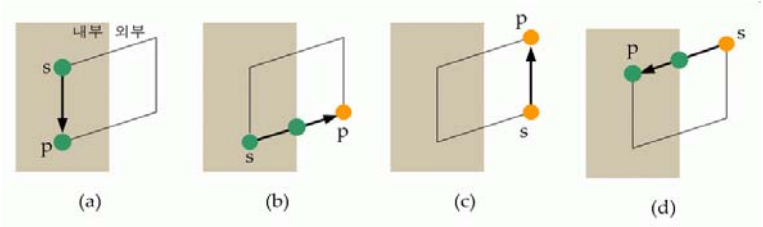
```
void clipLine (dcPt winMin, dcPt winMax, wcPt2 p1, wcPt2 p2)
{
  float u1 = 0.0, u2 = 1.0, dx = p2.x - p1.x, dy;

  if (clipTest (-dx, p1.x - winMin.x, &u1, &u2))
    if (clipTest (dx, winMax.x - p1.x, &u1, &u2)) {
      dy = p2.y - p1.y;
      if (clipTest (-dy, p1.y - winMin.y, &u1, &u2))
        if (clipTest (dy, winMax.y - p1.y, &u1, &u2)) {
          if (u2 < 1.0) {
            p2.x = p1.x + u2 * dx;
            p2.y = p1.y + u2 * dy;
          }
          if (u1 > 0.0) {
            p1.x += u1 * dx;
            p1.y += u1 * dy;
          }
          lineDDA (ROUND(p1.x), ROUND(p1.y),
                   ROUND(p2.x), ROUND(p2.y));
        }
    }
}
```

서더런드 핫지먼 알고리즘

- 절단 다각형을 기준으로 순서대로 절단

[그림 8-23] 서더런드-핫지만 알고리즘의 처리순서

- 절단 규칙

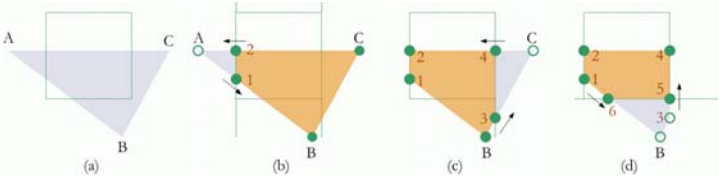


[그림 8-24] 서더런드-핫지먼 알고리즘의 절단규칙

32

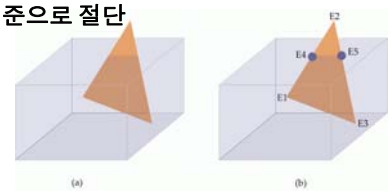
서더런드 핫지먼 알고리즘

- 선분을 연장한 직선을 기준으로 절단
- Ex. 좌변기준의 절단



[그림 8-25] 삼각형의 절단

- 3차원 절단
  - 상, 하, 좌, 우, 전, 후의 6개의 면을 기준으로 절단
  - 면을 기준으로 내외부 판정

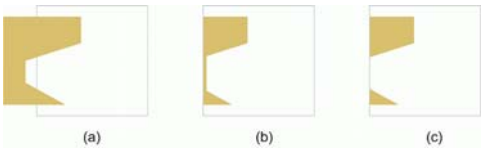


[그림 8-26] 3차원 서더런드-핫지먼 알고리즘

- Silicon Graphics의 Geometry Engine에 사용

볼록, 오목

- 서더런드-핫지먼
  - 볼록 다각형에만 적용
  - 하나의 다각형으로 취급
  - 오목 다각형 처리결과: 오류



[그림 8-28] 오목 다각형 처리결과

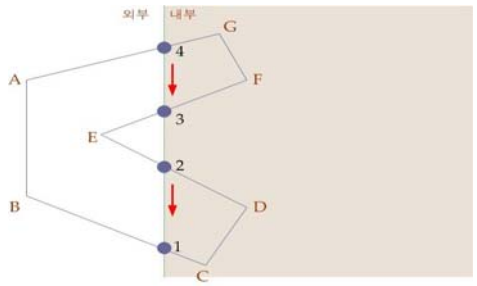
- 해법 1: 다각형 분할(Tessellation)
  - 오목 -> 볼록

[그림 8-29] 다각형 분할

- 해법 2: 웨일러-애서톤 알고리즘

웨일러-애서톤 알고리즘

- 내부에서 외부로 가는 교차점이 추가되면 즉시 그 교차점으로부터 절단 사각형을 따라서 반 시계 방향으로 간다. 즉, 가장 최근에 외부에서 내부로 들어온 교차점을 만날 때까지 간다.
- 1-C-D-2로 구성되는 하나의 다각형이 완성



[그림 8-30] 웨일러-애서톤 알고리즘

- 분리된 여러 개의 다각형을 생성함

35

- Sutherland & Hodgman Polygon Clipping

```
/* clipSuthHodge, Chapter 6, pp. 239-242 */

#include "graphics.h"

/* EXAMPLE STARTS HERE */
typedef enum { Left, Right, Bottom, Top } Edge;
#define N_EDGE 4

int inside (wcPt2 p, Edge b, dcPt wMin, dcPt wMax)
{
  switch (b) {
  case Left:   if (p.x < wMin.x) return (FALSE); break;
  case Right:  if (p.x > wMax.x) return (FALSE); break;
  case Bottom: if (p.y < wMin.y) return (FALSE); break;
  case Top:    if (p.y > wMax.y) return (FALSE); break;
  }
  return (TRUE);
}

int cross (wcPt2 p1, wcPt2 p2, Edge b, dcPt wMin, dcPt wMax)
{
  if (inside (p1, b, wMin, wMax) == inside (p2, b, wMin, wMax))
    return (FALSE);
  else return (TRUE);
}
```

```
wcPt2 intersect (wcPt2 p1, wcPt2 p2, Edge b,
                 dcPt wMin, dcPt wMax)
{
  wcPt2 iPt;
  float m;

  if (p1.x != p2.x) m = (p1.y - p2.y) / (p1.x - p2.x);
  switch (b) {
  case Left:   iPt.x = wMin.x;
               iPt.y = p2.y + (wMin.x - p2.x) * m;
               break;
  case Right:  iPt.x = wMax.x;
               iPt.y = p2.y + (wMax.x - p2.x) * m;
               break;
  case Bottom: iPt.y = wMin.y;
               if (p1.x != p2.x) iPt.x = p2.x + (wMin.y - p2.y) / m;
               else iPt.x = p2.x;
               break;
  case Top:    iPt.y = wMax.y;
               if (p1.x != p2.x) iPt.x = p2.x + (wMax.y - p2.y) / m;
               else iPt.x = p2.x;
               break;
  }
  return (iPt);
}
```

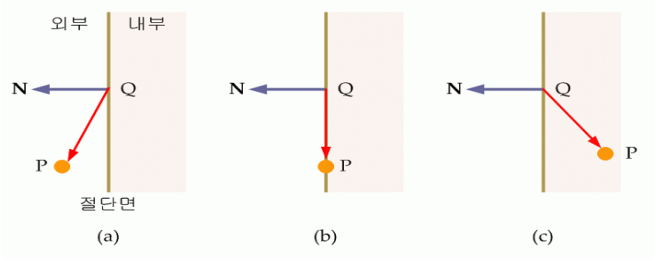
```
void clipPoint (wcPt2 p, Edge b, dcPt wMin, dcPt wMax,
                wcPt2 * pOut, int * cnt, wcPt2 * first[], wcPt2 * s)
{
  wcPt2 iPt;

  /* If no previous point exists for this edge, save this point. */
  if (!first[b]) first[b] = &p;
  else
    /* Previous point exists.  If 'p' and previous point cross edge,
       find intersection.  Clip against next boundary, if any.  If
       no more edges, add intersection to output list. */
    if (cross (p, s[b], b, wMin, wMax)) {
      iPt = intersect (p, s[b], b, wMin, wMax);
      if (b < Top)
          clipPoint (iPt, b+1, wMin, wMax, pOut, cnt, first, s);
      else {
          pOut[*cnt] = iPt;  (*cnt)++;
      }
    }
  s[b] = p;          /* Save 'p' as most recent point for this edge */

  /* For all, if point is 'inside' proceed to next clip edge, if any */
  if (inside (p, b, wMin, wMax))
    if (b < Top)
      clipPoint (p, b+1, wMin, wMax, pOut, cnt, first, s);
    else {
      pOut[*cnt] = p;  (*cnt)++;
    }
}
```

```
void closeClip (dcPt wMin, dcPt wMax, wcPt2 * pOut,
                int * cnt, wcPt2 * first[], wcPt2 * s)
{
  wcPt2 i;
  Edge b;
  for (b = Left; b <= Top; b++) {
    if (cross (s[b], *first[b], b, wMin, wMax)) {
      i = intersect (s[b], *first[b], b, wMin, wMax);
      if (b < Top)
        clipPoint (i, b+1, wMin, wMax, pOut, cnt, first, s);
      else
        pOut[*cnt] = i;  (*cnt)++;
    }
  }
}
int clipPolygon (dcPt wMin, dcPt wMax, int n,
                 wcPt2 * pIn, wcPt2 * pOut)
{
  /* 'first' holds pointer to first point processed against
     a clip edge.  's' holds most recent point processed
     against an edge */
  wcPt2 * first[N_EDGE] = { 0, 0, 0, 0 }, s[N_EDGE];
  int i, cnt = 0;

  for (i=0; i<n; i++)
    clipPoint (pIn[i], Left, wMin, wMax, pOut, &cnt, first, s);
  closeClip (wMin, wMax, pOut, &cnt, first, s);
  return (cnt);
}
```

```
#define N_PTS 6

void main (int argc, char ** argv)
{
  wcPt2 pts[N_PTS] =
    { 60, 20, 375, 80, 280, 280, 100, 280, 100, 100, 60, 20 };
  dcPt winMin = { 50, 50 };
  dcPt winMax = { 350, 250 };
  wcPt2 winPts[5];
  int i, nPts;
  long windowID = openGraphics (*argv, 400, 300);
  wcPt2 clippedPts[256];

  setBackground (WHITE);

  /* For illustration, draw the clipping rectangle */
  winPts[0].x = winMin.x; winPts[0].y = winMin.y;
  winPts[1].x = winMax.x; winPts[1].y = winMin.y;
  winPts[2].x = winMax.x; winPts[2].y = winMax.y;
  winPts[3].x = winMin.x; winPts[3].y = winMax.y;
  winPts[4].x = winMin.x; winPts[4].y = winMin.y;
  setColor (BLACK);
  pPolyline (5, winPts);
```

```
  /* For illustration, draw the complete set of lines */
  setColor (GREEN);
  pPolyline (N_PTS, pts);

  /* Clip the polygon against the window, returning 'clippedPts' */
  nPts = clipPolygon (winMin, winMax, N_PTS, pts, clippedPts);

  /* Repeat first point and draw closed, clipped polygon in red */
  clippedPts[nPts++] = clippedPts[0];
  setColor (RED);
  pPolyline (nPts, clippedPts);

  sleep (10);
  closeGraphics (windowID);
}
```

정점의 내외부 판정



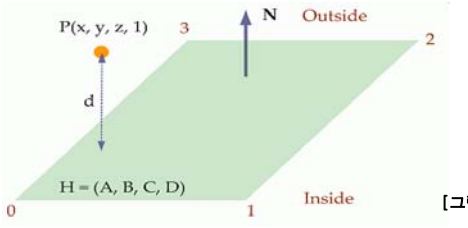
[그림 8-31] 정점의 내외부 판정

$$(P-Q)\cdot N>0 \; iff \; P is \; Outside \; the \; Clip \; Plane$$
$$(P-Q)\cdot N=0 \; iff \; P is \; on \; the \; Clip \; Plane$$
$$(P-Q)\cdot N<0 \; iff \; P is \; Inside \; the \; Clip \; Plane$$

39

동차좌표 사용

- 점과 평면간의 거리
  - 법선벡터 방향이 면의 외부로 정의됨



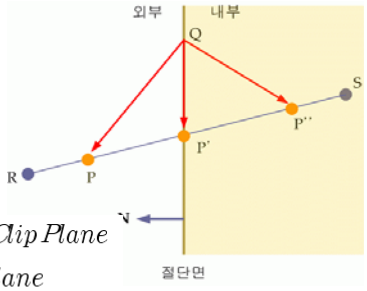
[그림 8-32] 점과 평면의 거리

$$d = H\cdot P = (A,B,C,D)\cdot(x,y,z,1) = Ax+By+Cz+D$$
$$Ax+By+Cz+D>0 \; iff \; P is \; Outside \; the \; Clip \; Plane$$
$$Ax+By+Cz+D=0 \; iff \; P is \; on \; the \; Clip \; Plane$$
$$Ax+By+Cz+D<0 \; iff \; P is \; Inside \; the \; Clip \; Plane$$

40

교차점 계산

$$p(t) = R + t(S - R) = (1-t)R + tS$$
$$x(t) = (1-t)R_x + tS_x$$
$$y(t) = (1-t)R_y + tS_y$$
$$z(t) = (1-t)R_z + tS_z$$



[그림 8-35] 교차점 계산

$$(p(t) - Q) \cdot N > 0 \; iff \; P is \; Outside \; the \; Clip \; Plane$$
$$(p(t) - Q) \cdot N = 0 \; iff \; P is \; on \; the \; Clip \; Plane$$
$$(p(t) - Q) \cdot N < 0 \; iff \; P is \; Inside \; the \; Clip \; Plane$$

$$p(t) \cdot N = Q \cdot N$$
$$(R + t(S - R)) \cdot N = Q \cdot N$$
$$t = (Q - R) \cdot N / (S - R) \cdot N$$

41

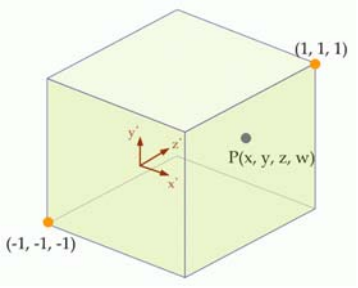
8.4 지엘의 절단–지엘의 절단

- **3차원좌표(x', y', z')**

$$-1 \leqq x' \leqq 1, -1 \leqq y' \leqq 1, -1 \leqq z' \leqq 1$$

- **정규화 장치좌표계**

$$-1 \leqq x/w \leqq 1, -1 \leqq y/w \leqq 1, -1 \leqq z/w \leqq 1$$



[그림 8-38] 정규화 부피

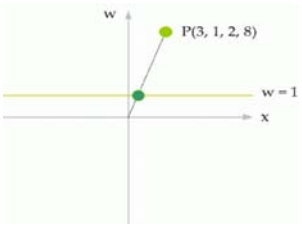
- **절단 좌표계 (동차 좌표)**

$$Case \; w > 0 : -w \leqq x \leqq w, -w \leqq y \leqq w, -w \leqq z \leqq w$$
$$Case \; w < 0 : -w \geqq x \geqq w, -w \geqq y \geqq w, -w \geqq z \geqq w$$

42

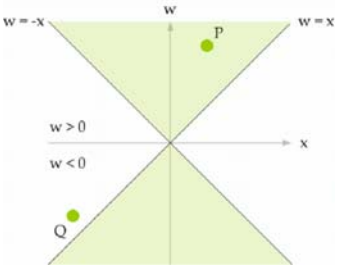
지엘의 절단

- 서더런드 핫지만 알고리즘과 유사
  - 지엘은 4차원 절단
  - 4차원 교차점 계산이 필요

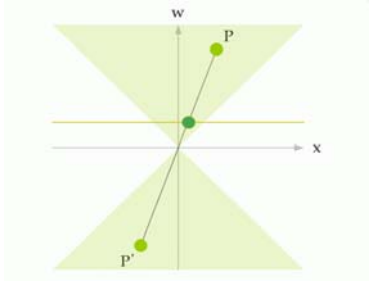


[그림 8-39] 동차좌표, 3차원 좌표

- 내부점, 외부점, 동일점



[그림 8-40] 내부점, 외부점



[그림 8-41] 동일점

추가적인 절단면

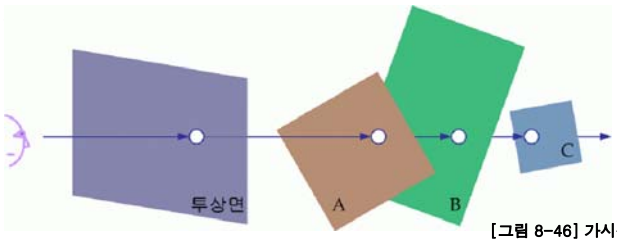
- **void glGetIntegerv (**GL_MAX_CLIP_PLANES**, &num)**

- **glClipPlane(GLenum, GLdouble *eq);**
  - GL_CLIP_PLANE0, …GL_CLIP_PLANE5
  - Ax+By+Cz+D=0 ➔ (A,B,C,D)

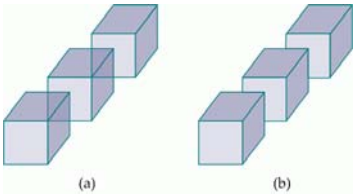
  - glEnable (GL_CLIP_PLANE0);
  - glDisable (GL_CLIP_PLANE0);

8.5 후면제거–은면제거

- **Hidden Surface Removal**
  - 앞 물체에 가려서 안 보이는 부분



[그림 8-46] 가시성

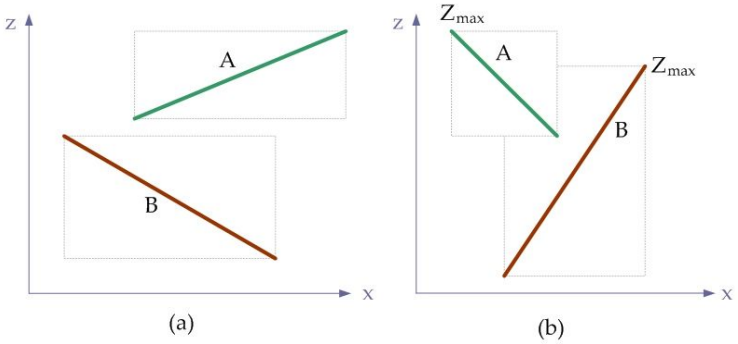


[그림 8-47] 은면 I

[그림 8-48] 은면 II

페인터 알고리즘

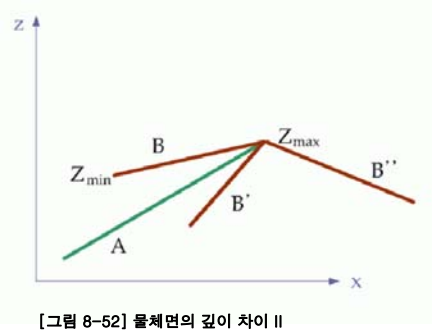
- 멀리 있는 배경위에 가까운 물체를 덧칠
  - 깊이 정렬(Depth Sort)이 필요
  - Zmax를 기준으로 물체를 정렬



[그림 8-51] 물체면의 깊이 차이 I

페인터 알고리즘

- B‘, B''
  - Zmin이 A의 Zmin보다 앞에 있으면 그것을 나중에 그려야 함.
- B
  - Zmin이 A의 Zmin보다 뒤에 있으면 그것을 먼저 그려야 함.
- B''
  - x 또는 y 범위가 서로 중첩되지 않으므로 어느 것을 먼저 그리던지 무관함.

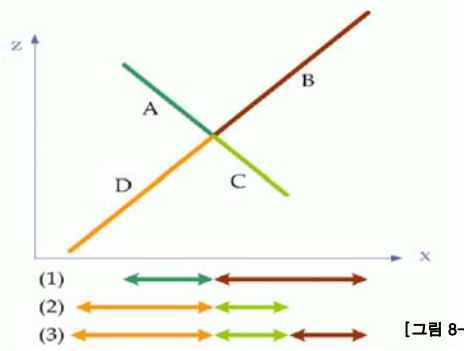


[그림 8-52] 물체면의 깊이 차이 II

47

페인터 알고리즘

- 면의 분할
- (1)
  - 먼저 A, B를 Zmax 기준으로 그려냄.
- (2)
  - C와 D는 Zmax는 같지만 x(또는 y)의 범위가 중첩되지 않으니 어느 것을 먼저 그려도 무방함
- (3)
  - 최종결과

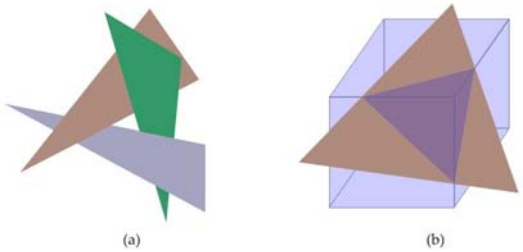


[그림 8-54] 면의 분할

48

페인터 알고리즘

- 가시성 사이클(Visibility Cycle)과 침투(Penetration)



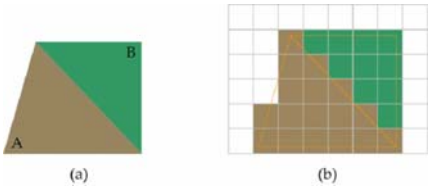
[그림 8-55] 사이클과 침투

- 페인터 알고리즘
  - 물체공간 알고리즘(Object Space Algorithm)
    - 정밀도는 높지만 실행속도가 느림
  - 경우에 따른 처리가 매우 복잡함.

49

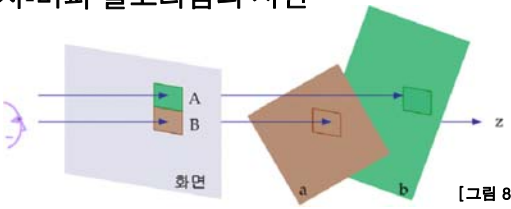
지-버퍼 알고리즘

- 물체공간 vs. 화소공간
  - 결국 화소공간으로 사상
  - 화소공간 해상도로 은면을 판단하면 됨



[그림 8-56] 물체공간, 화소공간

- 지-버퍼 알고리즘의 시선



[그림 8-57] 지-버퍼 알고리즘의 시선

50

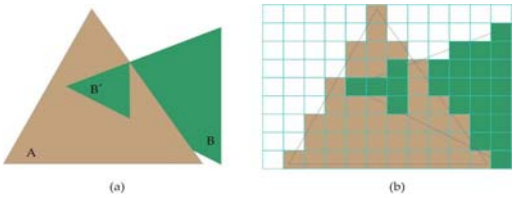
지-버퍼 알고리즘

- 지-버퍼(Z-Buffer) 또는 깊이버퍼(Depth Buffer)
- 지-버퍼 알고리즘

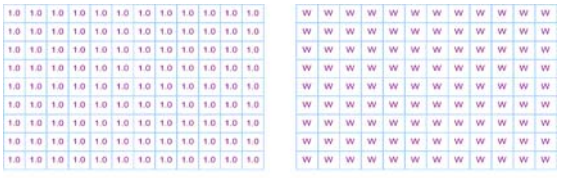
```
Initialize Frame Buffer with Background Color;
Initialize Z Buffer with Infinite Distance;
for Each Polygon {
    for Each Pixel {
      Calculate z of Intersection
      if (Calculated z < Current z of Z-Buffer) {
         Update Z-Buffer with Calculate z;
         Update Frame Buffer with the Color of Current Polygon;
       }
    }
}
```

51

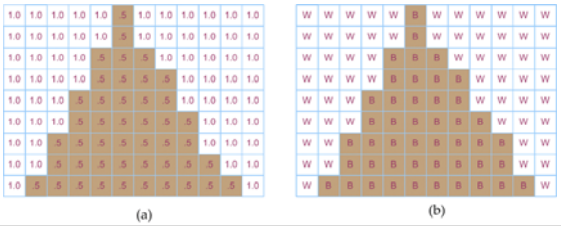
지-버퍼 알고리즘



[그림 8-58] 지-버퍼 알고리즘



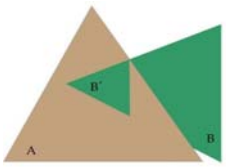
[그림 8-59] 지-버퍼의 초기화



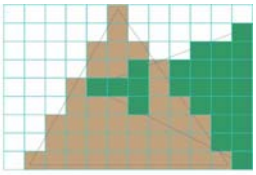
[그림 8-60] 갈색 삼각형 처리결과

52

지-버퍼 알고리즘

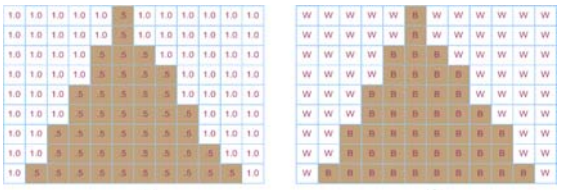


(a)



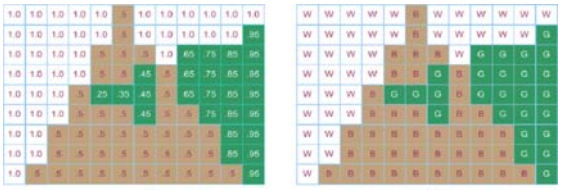
(b)

[그림 8-58] 지-버퍼 알고리즘



(a) (b)

[그림 8-60] 갈색 삼각형 처리결과



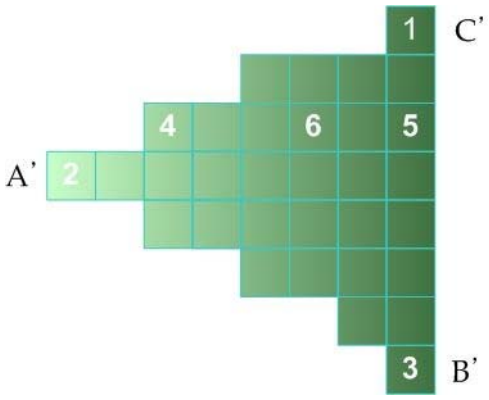
(a) (b)

[그림 8-61] 녹색 삼각형 처리결과

53

선형보간

- 래스터 변환 단계에서 실행
  - 정점으로부터 내부점의 깊이(Depth)및 컬러(Color)를 보간



[그림 8-63] 선형보간

54

GL의 Z-Buffer

- **void glGetIntegerv (Glenum pname, Glint *params)**
  - GL_DEPTH_BITS, GL_RED_BITS, …
- **glutInitDisplayMode (unsigned int mode);**
  - glutInitDisplayMode (GLUT_DEPTH | GLUT_RGBA);
- **glEnable (GL_DEPTH_TEST);**
- **glDisable (GL_DEPTH_TEST);**
- **glClear(GL_DEPTH_BUFFER_BIT);**
  - glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
- **glClearDepth(1.0);**
- **glDepthFunc(Glenum func);**
  - GL_NEVER, GL_ALWAYS, GL_LESS, GL_LEQUAL, GL_EQUAL,
  - GL_GEQUAL, GL_NOTEQUAL
- **glDepthMask(GLboolean flag);**

55

10월 26일 실습문제

- 지난주에 구현했던 Robot 과제를 Upgrade 함
  - WireRobot을 이용해 SolidRobot 만들기
  - 조명과 표면 재질 등을 설정 : 404쪽의 InitLight() 참조
  - 카메라 모션에서 Double Buffering 사용하기
  - Back face Culling 사용: Keyboard를 사용해서 On/off 기능
  - CW, CCW로 표면 지정해 보기: glFrontFace()
  - Hidden Surface 제거하기
  - 전면과 후면 표시방법 조절해 보기: glPolygonMode()
    - GL_FRONT/GL_BACK, GL_FILL/ GL_LINE/ GL_POINT
  - 시스템의 상태 알아보기
    - void glGetIntegerv (Glenum pname, Glint *params)
      - GL_MAX_CLIP_PLANES
      - GL_DEPTH_BITS, GL_RED_BITS, …
  - Robot에 재미있는 동작 추가하기

56