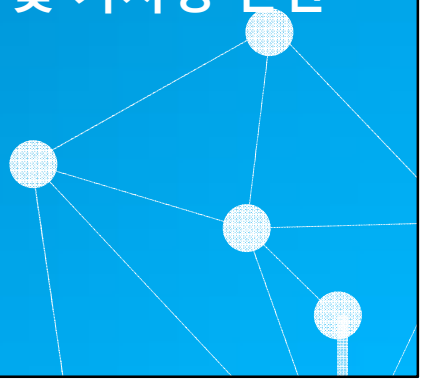
08 CHAPTER

절단 및 가시성 판단



8장. 학습 내용

- 벡터와 관련된 간단한 수학
- 후면 제거
- 절단 알고리즘
- 은면 제거 알고리즘

2

8.1 벡터와 관련된 간단한 수학

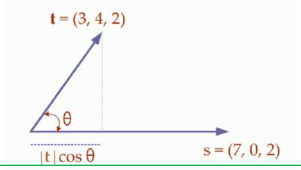
- 3차원 공간에서 평면의 표현
- 정점의 내외부 판정 방법
- 정점과 평면과의 거리
- 선분과 평면과의 교차점

3

복습: 벡터의 내적과 외적

복습: 벡터의 내적

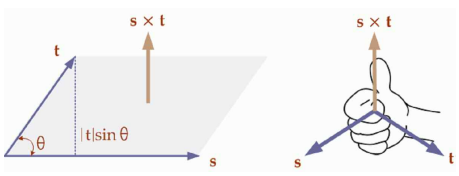
$$\boldsymbol{s} \cdot \boldsymbol{t} = |\boldsymbol{s}||\boldsymbol{t}|\cos\theta = s_x t_x + s_y t_y + s_z t_z$$



복습: 벡터의 외적

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

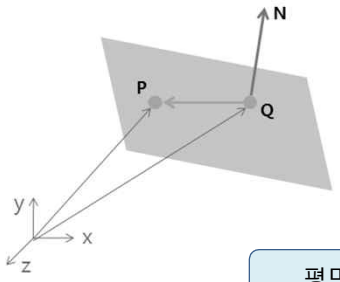
$$\boldsymbol{s} \times \boldsymbol{t} = -\boldsymbol{t} \times \boldsymbol{s}$$



4

평면의 표현

- P와 Q를 평면상의 임의의 점이라고 하고, N을 그 평면의 법선 벡터라고 하면 평면은 벡터의 내적을 이용하여 다음과 같이 정의됨



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

평면의 식

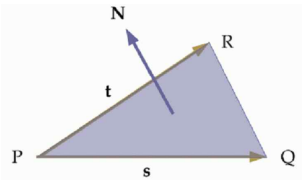
이 평면의 법선벡터 (A, B, C)

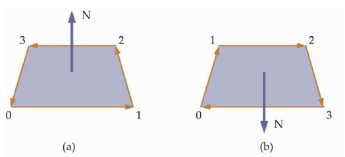
- 예) 법선벡터 (A,B,C)를 알고 평면상의 한 점(x,y,z)를 알고 있을 때 평면의 방정식은? D만 구하면 됨.

법선 벡터 구하기

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

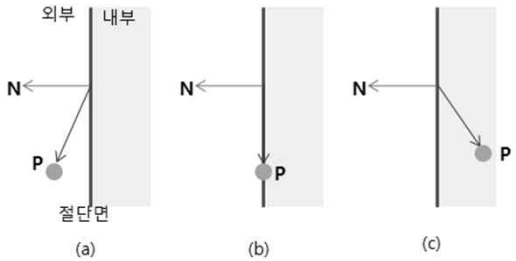
$$s=(Q_x-P_x,Q_y-P_y,Q_z-P_z)$$
$$t=(R_x-P_x,R_y-P_y,R_z-P_z)$$
$$N=s\times t$$





정점의 내부/외부 판정

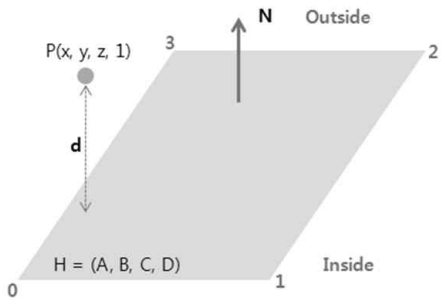
- 3차원 공간에서 어떤 면(절단면)을 기준으로 임의의 점이 내부에 있는지 외에 있는지를 결정 → 절단
- 내/외부 판정 방법 → 벡터의 내적
  - "내부"는 법선벡터 방향의 반대 → $(P-Q) \cdot N < 0$
  - "외부"는 법선벡터 방향 → $(P-Q) \cdot N > 0$
  - 면 상에 있는 점: $(P-Q) \cdot N = 0$



정점과 평면과의 거리

- 평면 H는 동차좌표(A,B,C,D)
- 법선벡터 N은 (A,B,C)

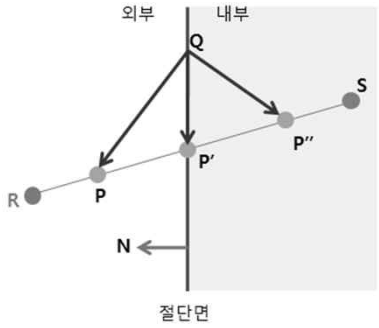
$$d = H \cdot P = (A,B,C,D) \cdot (x,y,z,1) = Ax + By + Cz + D$$



선분과 평면과의 교차점 X 생략

- 선분의 매개변수 방정식 표현

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



- 교차점 계산

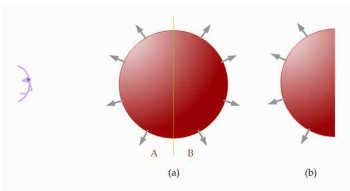
$$(p(t)-Q) \cdot N = 0$$
$$p(t) \cdot N = Q \cdot N$$
$$(R+t(S-R)) \cdot N = Q \cdot N$$
$$t = (Q-R) \cdot N / (S-R) \cdot N$$

- 만약 t<0 또는 t>1 → 선분과 평면은 만나지 않음

9

8.2 후면 제거

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



- 후면제거 개념
- OpenGL의 후면제거 함수

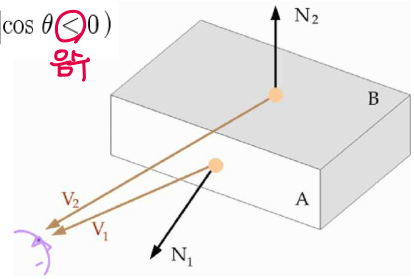
10

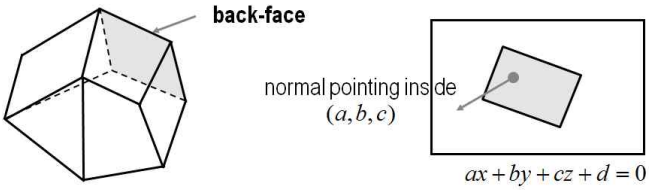
후면제거 개념 그림그리고 설명하기(10점)

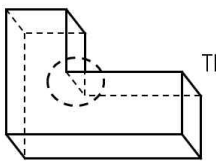
- 후면제거(Backface Culling, Backface Removal)
  - 시점과 면의 오리엔테이션만으로 판단
  - 보이지 않는 면의 거의 절반을 제거

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

내적 음수







The back-face culling does not work in general

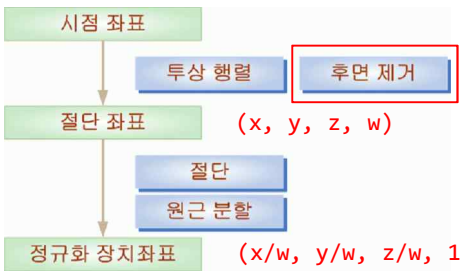
OpenGL 후면제거

- 후면제거 관련 함수들

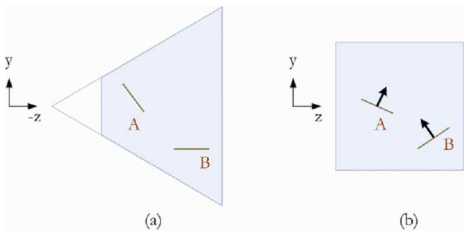
```
glEnable(GL_CULL_FACE);
glCullFace(GL_FRONT);
glDisable(GL_CULL_FACE);
```

- 데이터의 거의 절반을 줄여줌 → 3차원 그래픽스 파이프라인에서 가능한 한 빨리 처리해 주는 것이 유리
- OpenGL에서는
  - 시점좌표로 변환된 물체를 절단좌표로 변환하기 전에 후면제거
  - 절단이나 원근 분할 등 이후의 연산의 데이터를 줄임.

13



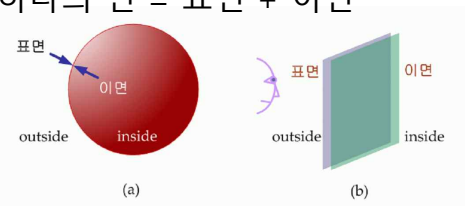
- 정규화 가시부피
  - 법선벡터의 z 값만으로 판단가능



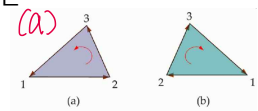
14

표면과 이면

- 하나의 면 = 표면 + 이면

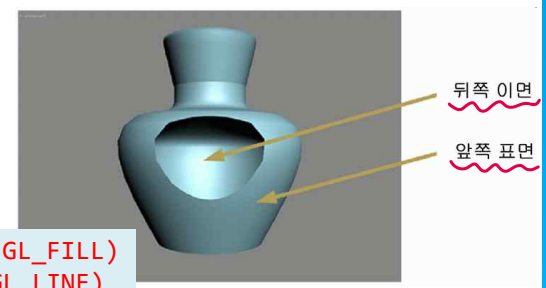


- 표면
  - 시계방향으로 정의된 면 (b)
    - glFrontFace(GL_CCW)
  - 반시계방향으로 정의된 면
    - glFrontFace(GL_CW)



15

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




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

16

- [Lab 8-1] 전면과 후면의 이면을 점이나 선, 채움으로 그릴 수 있도록 메뉴를 추가하라.

17

8.3 절단 알고리즘

- 절단의 의미와 다양한 객체의 절단
- Cohen-Sutherland Line Clipping Algorithm
- Liang & Barsky's Line Clipping Algorithm
- 다각형의 절단과 텍스트의 절단

18

절단(Clipping) 이란?

- 장면(scene)을 이루는 각 그래픽 요소 또는 일부분들이 가시부피 안에 들어가는지를 판단하는 과정

- 응용분야
  - 관측(viewing)을 위해 정의된 장면의 일부를 추출함
  - 3차원 관측에서 보여지는(visible) 표면의 결정
  - 선분이나 물체 경계의 앤티엘리어싱
  - 다중 윈도 환경에서의 화면 출력
  - 복사, 이동, 삭제, 중복 등을 위해 그림의 일부를 그리는 경우

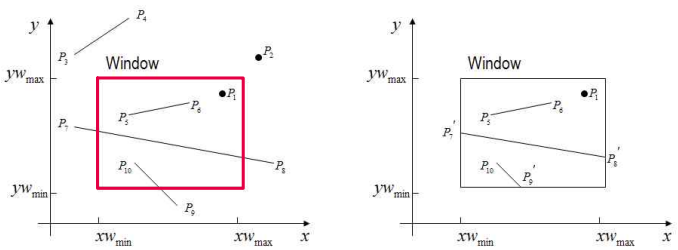
19

절단의 종류

- 종류: 점, 선분, 영역(다각형), 곡선, 텍스트 등
- 점 절단

$$x_{\min} \le x \le x_{\max}$$
$$y_{\min} \le y \le y_{\max}$$

- 선 절단

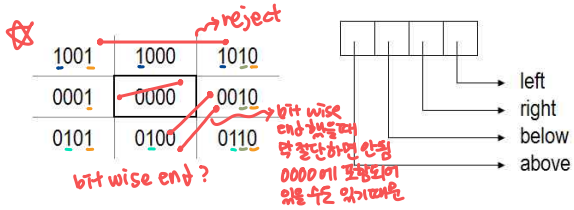


  - Cohen-Sutherland 알고리즘
  - Liang과 Barsky 알고리즘

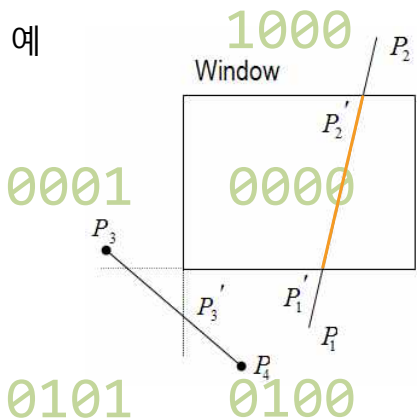
20

Cohen-Sutherland Line Clipping Algorithm

- 기본 아이디어
  - Encode the line endpoints
  - Successively divide the line segments so that they are completely contained in the window or completely lies outside window
  - Division occurs at the boundary of window
- 평면의 9 분할 (2차원에서의 절단의 경우)



- 윈도우와 선분의 관계 예
  - P1-P2=0100-1000
  - P3-P4=0001-0100



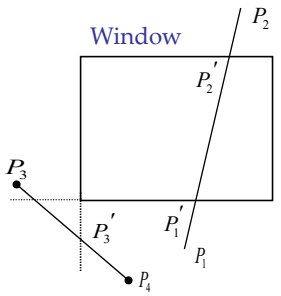
- accept와 reject 조건을 검사하며, 교차점 계산

  - Accept 조건: (Code1 & Code2) == 0000
  - Reject 조건: 1 at the same bits : reject the line
    ex) 1001 - 0101 (c1 & c2) != 0000
  - 교차점의 계산: Line Equation 이용

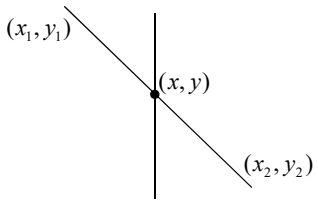
$$y = y_1 + m(x - x_1) \quad \text{수직 경계와의 교차}$$

$$x = x_1 + \frac{y - y_1}{m} \quad \text{수평 경계와의 교차}$$

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



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



교차점: Line Equation이용

$$y = y_1 + m(x - x_1) \quad \text{수직 경계와의 교차}$$

$$x = x_1 + \frac{y - y_1}{m} \quad \text{수평 경계와의 교차}$$

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

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

25

```
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
              …
        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
        …
  end {accept}

begin {clipALine}
    done = false;
    display = false;
    while not done do begin
          encode (x1, y1, code1);
          encode (x2, y2, code2);
          ① if accept(code1, code2) { done=true; display=true; }
          ② else if reject (code1, code2) 둘다 외부
                        { done=false; display=false; }
          ③ else { 둘중 하나는 외부
              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}
```

26

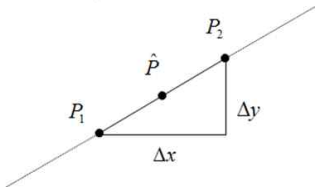
- [Lab 8-2] Cohen-Sutherland 선분 절단 알고리즘을 3차원 공간으로 확장하는 방법을 설명하라. 프로그램 8.1을 확장하여 3차원 절단 함수를 구현해 보라.

27

Liang & Barsky's Line Clipping Algorithm

- 선분의 클리핑을 위해 매개변수 방정식을 이용
- 선분의 매개변수 방정식 표현(평면)

$$\hat{P} = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 + u(x_2 - x_1) \\ y_1 + u(y_2 - y_1) \end{pmatrix} = \begin{pmatrix} x_1 + u\triangle x \\ y_1 + u\triangle y \end{pmatrix}, \quad 0 \le u \le 1$$



- 절단 알고리즘
  - u1, u2를 찾음

$$x_{\min} \le x_1 + u\triangle x \le x_{\max}$$
$$y_{\min} \le y_1 + u\triangle y \le y_{\max}$$

28

- 네 개의 부등식 정형화

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

$$\therefore x = x_1 + u\Delta x$$
$$y = y_1 + u\Delta y$$

$$x_{\min} \le x_1 + u\Delta x \le x_{\max}$$
$$y_{\min} \le y_1 + u\Delta y \le y_{\max}$$
$$\downarrow$$
$$u(-\Delta x) \le x_1 - x_{\min}$$
$$u\overline{P}_1 \le \overline{q}_1$$

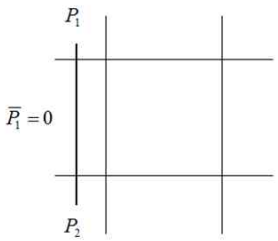
$$u\overline{P}_k \le \overline{q}_k, \quad k = 1,2,3,4$$

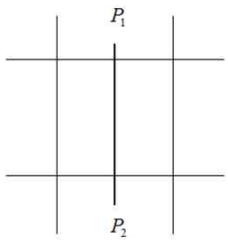
$$\overline{P}_1 = -\Delta x, \overline{q}_1 = x_1 - x_{\min}$$
$$\overline{P}_2 = \Delta x, \overline{q}_2 = x_{\max} - x_1$$
$$\overline{P}_3 = -\Delta y, \overline{q}_3 = y_1 - y_{\min}$$
$$\overline{P}_4 = \Delta y, \overline{q}_4 = y_{\max} - y_1$$

$\overline{P}_k = 0$ ($\Delta x = 0 \Rightarrow$ 수직선, $\Delta y = 0 \Rightarrow$ 수평선)
$\overline{P}_k < 0$ 3 cases
$\overline{P}_k > 0$

(1) $\overline{P}_k = 0$: 평행선의 경우

그림과 같이 수평선이나 수직선의 경우이다. 이 경우는 $\overline{q}_k < 0$이면 이 직선이 해당 경계 밖에 있는 것이므로 reject한다. 그렇지 않으면 계속 처리한다.



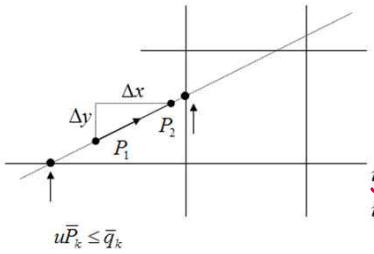


$\overline{q}_1 < 0$: reject $\overrightarrow{P_1P_2}$ (절단)

$\overline{q}_1 \ge 0$: need further processing

(2) $\overline{P_k} < 0$: 선분이 외부에서 내부로 들어가는 경우

이 경우는 그림과 같이 P1(매개변수가 u_1인 점)이 외부에 있는 경우이다. 따라서 u_1이 갱신되어야 하는데, 해당 경계선을 이용해 u_1을 갱신한다. 만약 u_1이 u_2보다 커지면 이 선분은 reject 된다. 그려질 부분이 남지 않았기 때문이다.



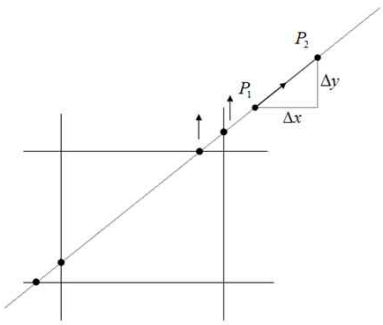
$u\overline{P_k} \le \overline{q}_k$

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

$\hat{u}_1 = \max\{0, u_k\}$ reject if $\hat{u}_1 > 1$!!!

(3) $\overline{P_k} > 0$: 선분이 내부에서 외부로 나가는 경우

P2(매개변수가 u_1인 점)가 외부에 있는 경우이다. u_2가 줄어들어야 하는데, 해당 경계선을 이용해 u_2를 갱신한 후 이 값이 u_1보다 작아지면 역시 reject 된다.



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

$\hat{u}_2 = \min\{1, u_k\}$ reject if $\hat{u}_2 < 0$!!!

```
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}
```

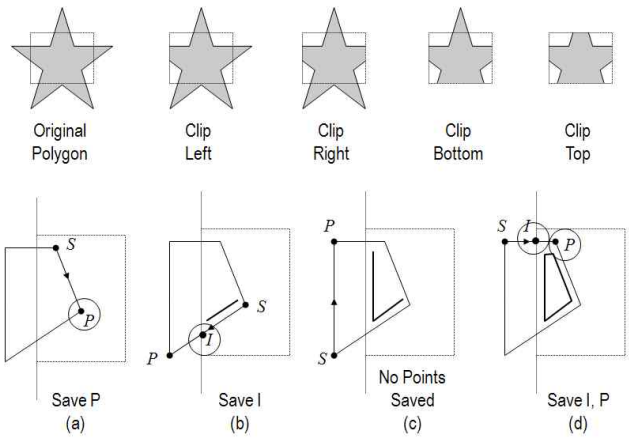
```
begin {clipLiangBarsky}
   u1 = 0;
   u2 = 1;
   dx = x2 - x1;
   dy = y2 - y1;

   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}
```

33

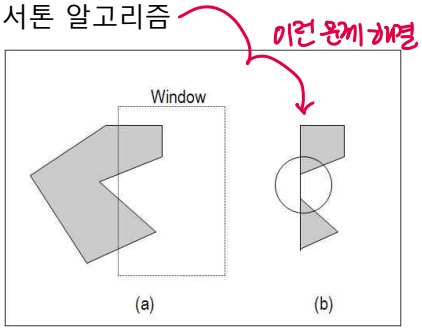
다각형의 절단

- Sutherland와 Hodgeman의 다각형 절단 알고리즘



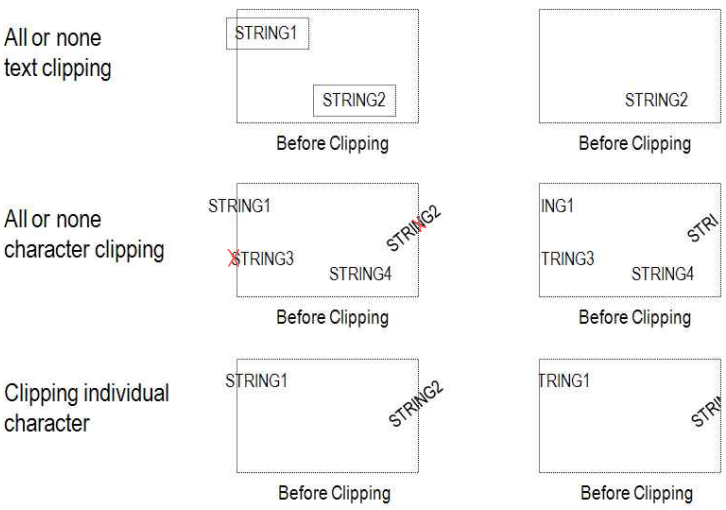
34

- 다각형에 오목한 부분이 있는 경우는 다음 그림과 같이 자연스럽지 못한 부분이 발생
  - 볼록 다각형으로 분할한 후 절단하는 방법
  - 웨일러-에서톤 알고리즘



35

텍스트 클리핑



36

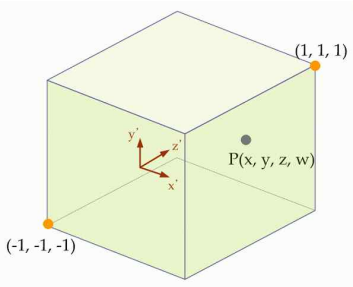
OpenGL의 절단

- 3차원좌표(x′, y′, z′)

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

- 정규화 장치좌표계

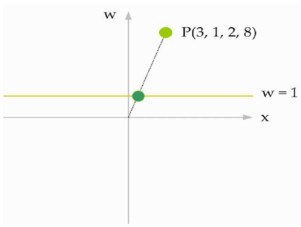
$$-1 \leqq x/w \leqq 1, -1 \leqq y/w \leqq 1, -1 \leqq z/w \leqq 1$$



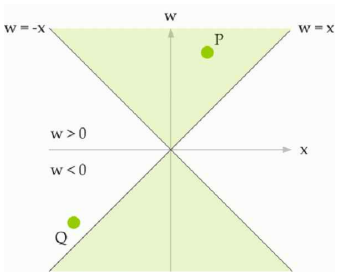
- 절단 좌표계 (동차 좌표)

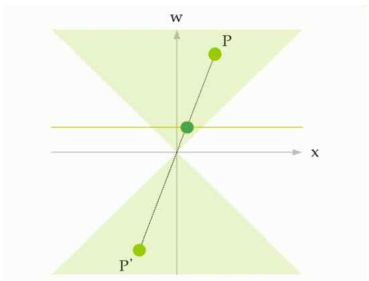
$$Case\ w > 0: -w \leqq x \leqq w, -w \leqq y \leqq w, -w \leqq z \leqq w$$
$$Case\ w < 0: -w \geqq x \geqq w, -w \geqq y \geqq w, -w \geqq z \geqq w$$

- 지엘은 4차원 절단
  - 4차원 교차점 계산이 필요



- 내부점, 외부점, 동일점





- 추가적인 절단면 8개 정도제공
- void glGetIntegerv (GL_MAX_CLIP_PLANES, &num)

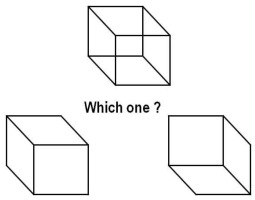
- glClipPlane(GLenum, GLdouble *eq);
  - GL_CLIP_PLANE0, …GL_CLIP_PLANE5
  - Ax+By+Cz+D=0 ➔ (A,B,C,D)

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

39

8.4 은면 제거 알고리즘

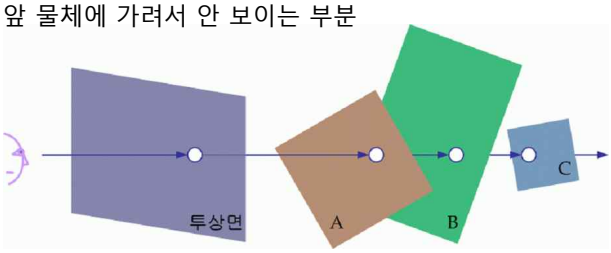
- 은면 제거의 개념 앞부→전면, 앞부→화면
- 객체 공간법(Object space method)
- 영상 공간법(Image space method)
- OpenGL의 Z-Buffer 함수

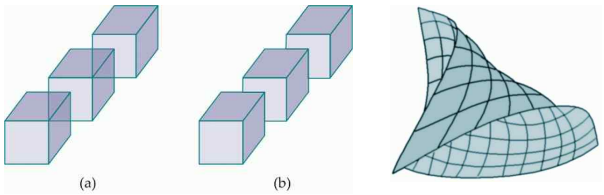


40

은면제거 개념

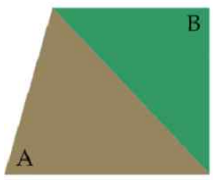
- Hidden Surface Removal
  - 앞 물체에 가려서 안 보이는 부분



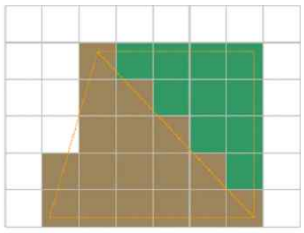


41

객체공간법 / 영상공간법



(a)

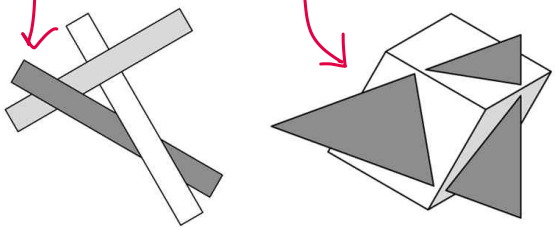


(b)

42

객체공간법(Object space method)

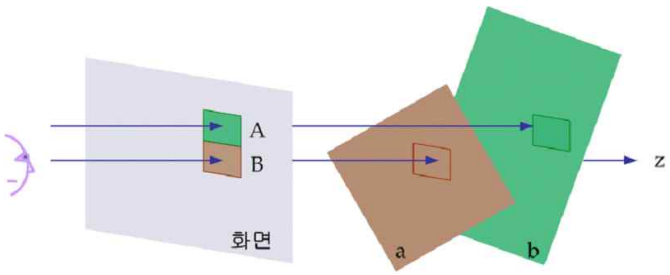
- Painter's algorithm
  - 모든 면들을 깊이 방향으로 정렬
  - 정렬 후 시점에서 가장 먼 면부터 순차적으로 출력
- 정밀도는 높지만 실행속도가 느림
- 경우에 따른 처리가 매우 복잡함
  - 가시성 사이클 및 표면의 침투 문제



43

영상공간법(Image space method)

- 투영된 픽셀 평면에서 객체가 보이는지 여부를 검사하는 방법 → Z-버퍼 알고리즘
- 지-버퍼 알고리즘의 시선



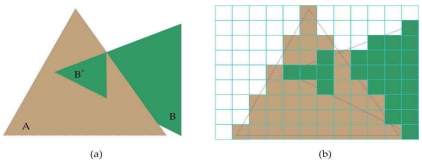
44

Z-버퍼 알고리즘

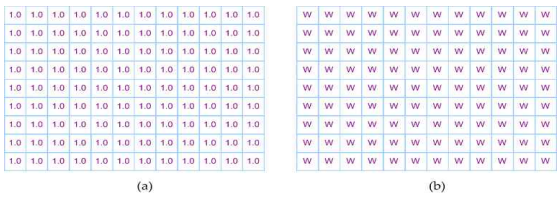
- 깊이 버퍼 알고리즘

```
Initialize Frame Buffer(F_Buffer) with Background Color;
Initialize Z_Buffer with Infinite Distance;
  for Each Polygon {
        or Each Pixel {
           calculate z of Intersection
           if (Calculated z < Current z of Z_Buffer) {
              update Z_Buffer with Calculate z;
              update F_Buffer with the Color of Current Polygon;
           }
        }
  }
```

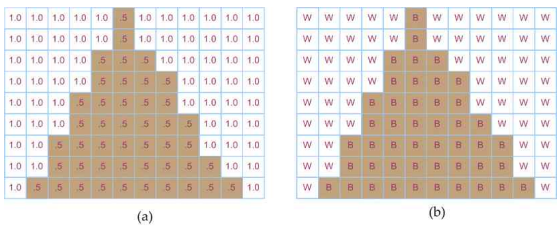

- 깊이 버퍼 알고리즘



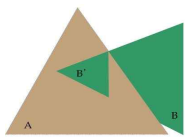
- 버퍼 초기화

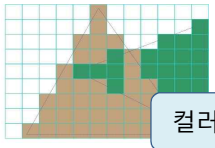


- 갈색 삼각형 처리 결과



- 깊이 버퍼 알고리즘





깊이 버퍼

컬러 버퍼

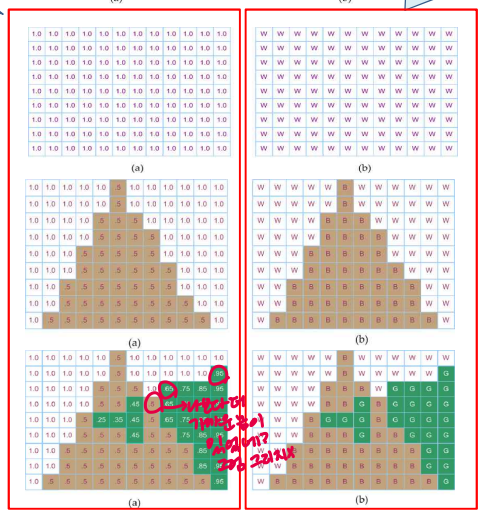
순서

① 버퍼 초기화
  - 깊이 버퍼
  - 컬러 버퍼

② 갈색 삼각형 처리

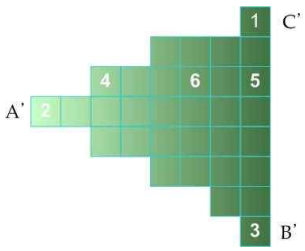
언제 그려졌지 상관X

③ 초록색 삼각형 처리



선형 보간

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



OpenGL의 Z-Buffer 함수

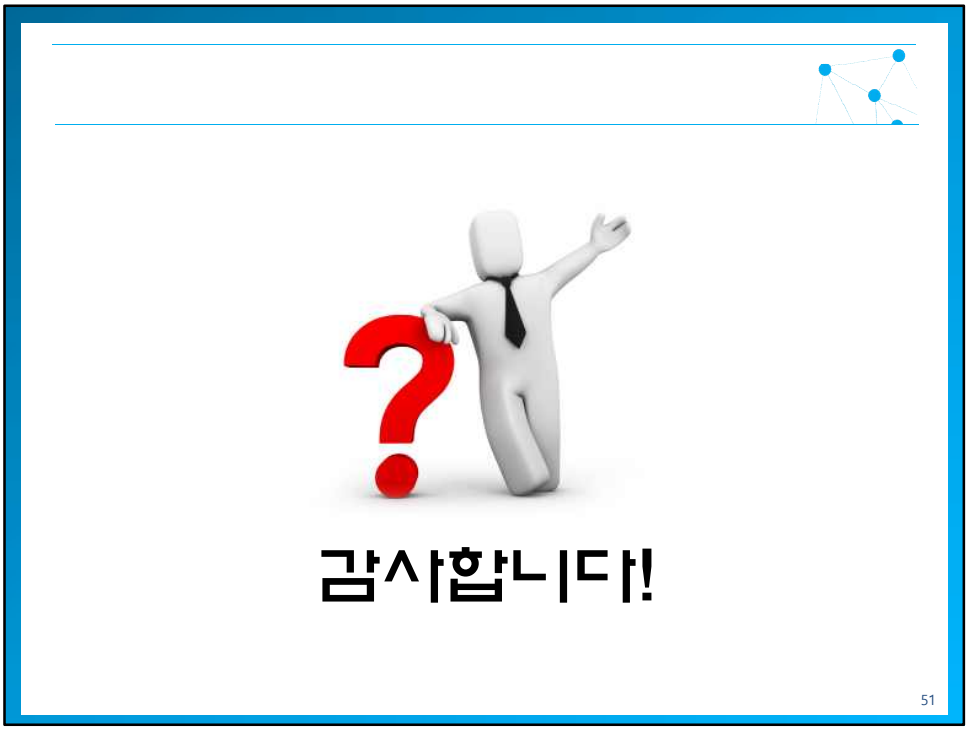
- void glGetIntegerv (Glenum pname, Glint *params)
  - GL_DEPTH_BITS, GL_RED_BITS, …
- glutInitDisplayMode (unsigned int mode);
  - glutInitDisplayMode (GLUT_DEPTH | GLUT_RGBA);
- glEnable (GL_DEPTH_TEST);
- glDisable (GL_DEPTH_TEST);
- glClear(GL_DEPTH_BUFFER_BIT);
  - glClear(GL_COLOR_BUFFER_BIT | 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);

49

8장 연습문제

- [Lab 8-4] 다음과 같은 클리핑 알고리즘 테스트를 위한 프로그램을 작성하시오.
  - - 화면 중앙에 사각형을 그린다. (클리핑 윈도우 영역. 고정되어도 됨)
  - - 마우스 이벤트를 처리하여 화면에 선분 하나를 그린다.
  - - 키보드 'c'를 누르면 입력한 선분을 코헨-서덜랜드 알고리즘이나 리앙-바스키 알고리즘을 이용해 클리핑 한 후 결과를 그린다.
- [Lab 8-5] 화면에 다각형을 그린 후 동일한 방법을 이용하여 클리핑한 결과를 출력하는 프로그램을 작성하시오.

50


감사합니다!
51