과제5 16011024 주수헌

-차이점과 비교점을 설명하기에 앞서 저는 코드 처음 실행을

GLboolean bBlend = false;

GLboolean bSmooth = false;

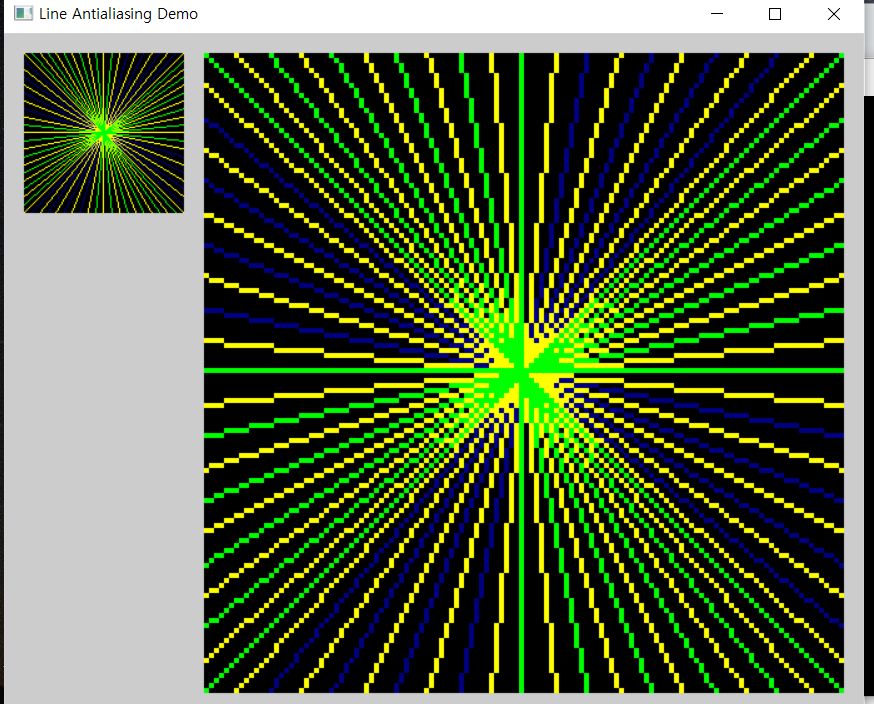
으로 놓아 혼합 함수와 스무딩 함수가 처음 실행화면에는 실행되지 않도록 했습니다.

1.혼합 함수 사용

-인접화소의 색을 주어진 화소에 스며들게 함으로써 에일리어싱을 감쇄합니다.

혼합함수를 사용함으로서 색이 겹치게 된 경우 이미 기록되어져 있는 값과 새로 그려지는 값의 논리 연산 방법을 지정하여 연산 결과를 넣어 효과를 넣습니다.

-초기 실행시 혼합함수만 사용했을 때 실행 화면



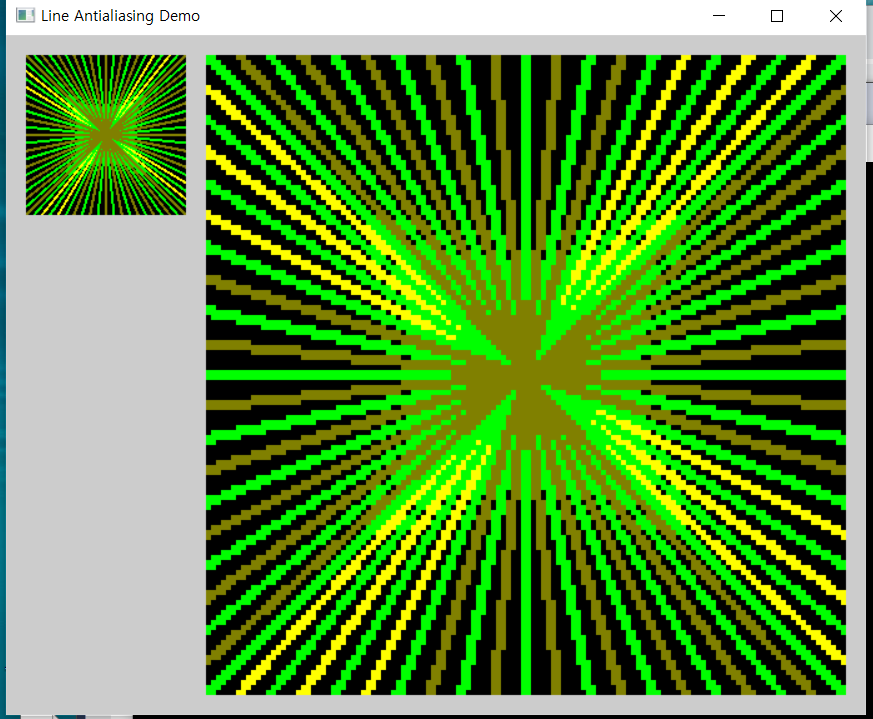
-실행 결과를 보면 혼합함수만 사용하 기존 실행화면과 차이가 없습니다. 혼합 함수 같은 경우는 위에서 설명했다싶이 색이 겹치는 경우, 이미 기록되어져 있는 값과 새로 그려지는 값의 논리 연산 방법을 지정하여 연산 결과를 넣어 효과를 넣는 것입니다.

2.스무딩 함수 사용

-에일리어싱 제거를 위해 두 색상의 경계면에 중간색을 삽입하는 함수입니다.

예제에서는 glEnable(GL\_LINE\_SMOOTH); 을 사용하였습니다.

-초기 실행시 스무딩 함수만 사용했을 때 실행결과입니다

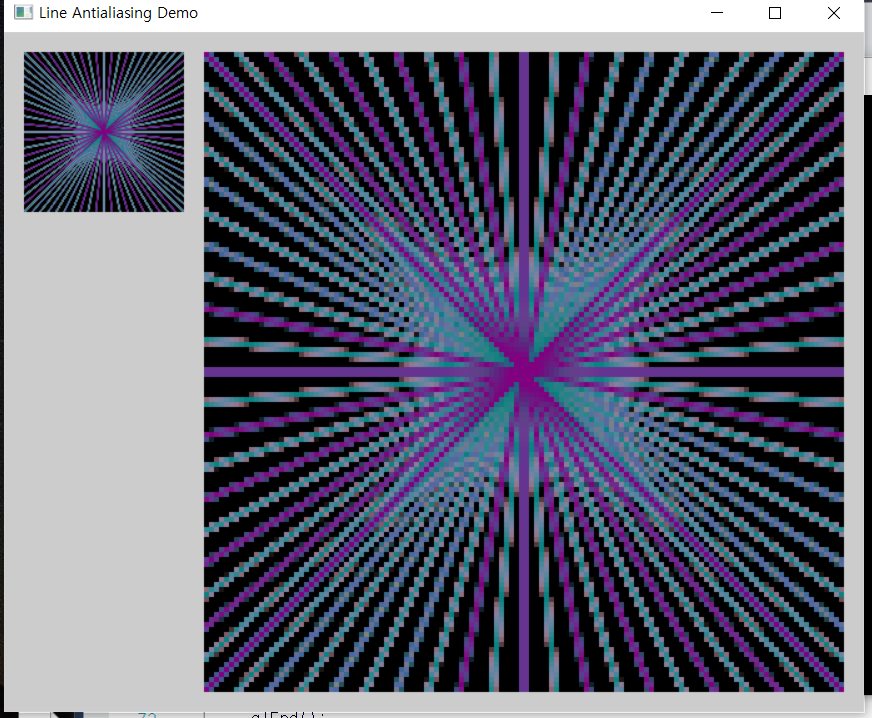
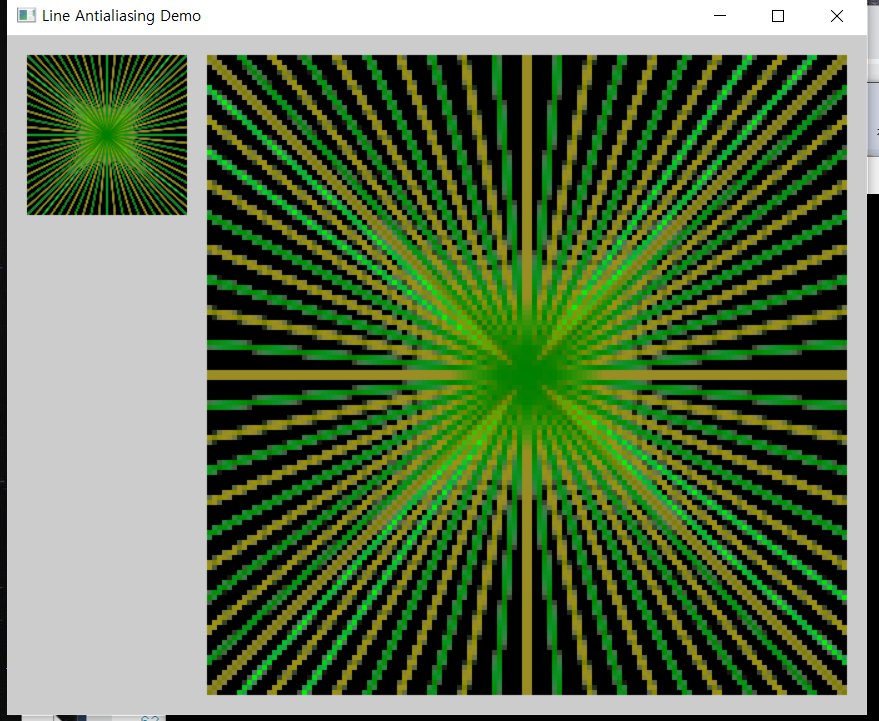


-스무딩 함수를 사용했을 경우 처음 실행화면보다 선이 굵어졌습니다. 경계면의 색상보완 때문에 그런 것입니다. 그러나 여기서 혼합 함수를 사용하지 않고 스무딩 함수만 사용하여 경계면에 혼합된 생각을 표현하지 못하였습니다.

3.두가지 함수 모두 사용

-두가지 함수를 모두 사용했을 경우 우리가 원하는 앤티 에일리어싱을 표현할 수 있었습니다. 경계면의 색상의 보안을 할 수 있으며 혼합함수 때문에 혼합된 색상을 표현할 수 있었습니다.

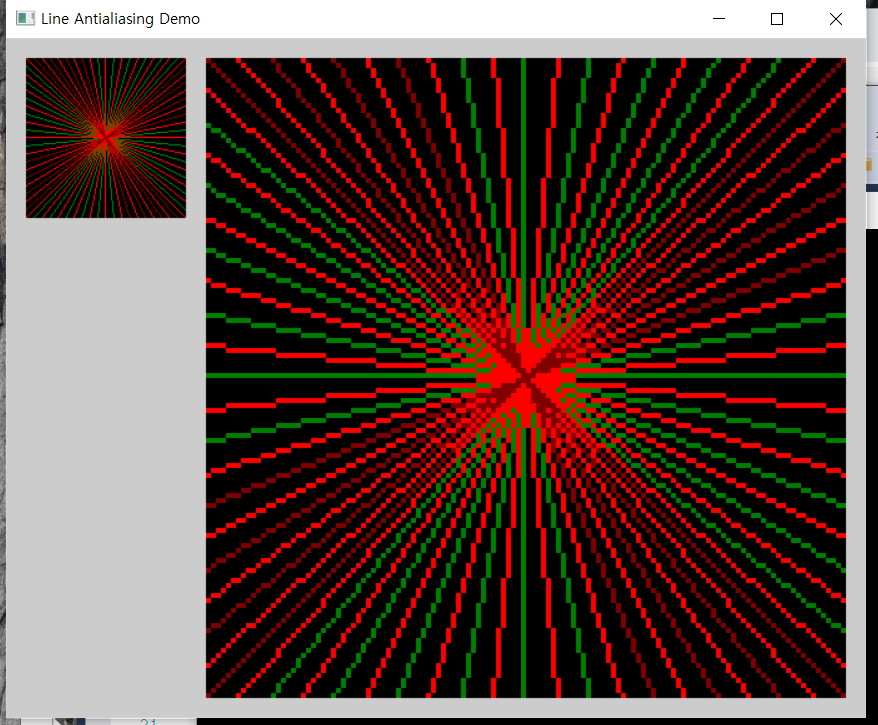
-다음은 두가지 함수를 모두 사용했을 때의 실행 결과화면입니다



4.두가지 함수 모두 사용안함

-두가지 함수 모두 사용을 안 했을 경우 그래픽 화면에서 에일이러싱 현상 모습을 그대로 나타남을 볼 수 있었습니다.

-아래는 두가지 함수 모두 사용하지 않을 경우의 실행화면입니다.



\*world\_display(), screen\_display() 분석

(1) world\_display()

-주석으로 함수내 각 코드를 분석 하였습니다.

void world\_display() {

Red = PALETTE[Index][0] / 255.0f; //어떤 인덱스든 빨간부분 나타냄

Green = PALETTE[Index][1] / 255.0f; //어떤 인덱스든 초록부분 나타냄

Blue = PALETTE[Index][2] / 255.0f; //어떤 인덱스든 파랑부분 나타냄

glColor3f(Red, Green, Blue); //위에 코드에서 정한 rgb 색상으로 한다

if (bBlend) { //혼합 함수: 인접화소의 색을 주어진 화소에 스며들게하여 에일리어싱감쇄

glEnable(GL\_BLEND); //혼합기능 활성화

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA); //혼합모드를 설정

}

//GL\_SRC\_ALPHA의 블렌딩 함수는 새로운 색상 요소에 0.4를 곱하고

//GL\_ONE\_MINUS\_SRC\_ALPHA의 블렌딩 함수는 기존 색상 요소에 1-0.4=0.6을 곱하여

//두 연산결과를 더하여 최종색상을 결정한다

if (bSmooth) {

glEnable(GL\_LINE\_SMOOTH); //부드러운 선 기능 활성화

glHint(GL\_LINE\_SMOOTH\_HINT, GL\_NICEST);//여기에 옵션 다른 것도 줄 수 있음

//glHint(GL\_LINE\_SMOOTH\_HINT, GL\_FASTEST);

//glHint(GL\_LINE\_SMOOTH\_HINT, GL\_DONT\_CARE);

//추가점수부분

}

//GL\_NICEST은 조금 느리더라도 질이 좋은 안티 에일리어싱을 가해줌

//GL\_FASTEST: 질이 떨어져도 가장 빠르게 그리기 원할시 사용한다

//GL\_DONT\_CARE: 디폴드 값이며 OpenGL이 알아서 속도와 품질을 결정하게 된다

glBegin(GL\_LINES);

glVertex3f(-1.0 + fDelta, 1.0, 0.0); //2개의 점을 이어 선 표시

glVertex3f(1.0 - fDelta, -1.0, 0.0);

glVertex3f(-1.0, -1.0 + fDelta, 0.0); //2개의 점을 이어 선 표시

glVertex3f(1.0, 1.0 - fDelta, 0.0);

glEnd();

//그리고 싶은 선을 다 그렸으면 스무딩함수와 혼합함수를 모두 disable한다

if (bSmooth)

glDisable(GL\_LINE\_SMOOTH);

if (bBlend)

glDisable(GL\_BLEND);

glutSwapBuffers();

}

(2) screen\_display()

-주석을 이용하여 각 코드에 대한 분석을 하였다.

void screen\_display() {

static unsigned char buffer[128 \* 128 \* 4 \* 2];

glutSetWindow(world); //현재 윈도우 설정

glReadPixels(0, 0, 128, 128, GL\_RGBA, GL\_UNSIGNED\_BYTE, buffer);

//gl ReadPixels()는 프레임 버퍼로부터 픽셀블럭을 읽어낸다

//즉, 작은 창에 그려진 프레임 버퍼 내용을 메모리로 복사하는 함수이다

//프레임 버퍼로부터 읽은 첫번째 픽셀의 윈도우 창좌표 0,0으로 구체화한다

//그 위치는 픽셀 사각형 좌하단 기준으로 한다

//픽셀 사각형 가로세로를 128,128로 정한다

//GL\_RGBA은 픽셀 데이터 포멧이다. GL\_UNSIGNED\_BYTEd은 픽셀 데이터 타입이다

//반환하는 픽셀 데이터: buffer

glutSetWindow(screen); //screen이라는 서브윈도우를 설정

glPixelZoom(4, 4); //픽셀 확대 비율을 정한다

//픽셀 쓰기 작업에 대한 x y 확대 비율을 정한다

glDrawPixels(128, 128, GL\_RGBA, GL\_UNSIGNED\_BYTE, buffer);

//이 함수에 의해 큰 창에 해당하는 프레임 버퍼로 기록된다

glPixelZoom(0, 0);

glutSwapBuffers();

}

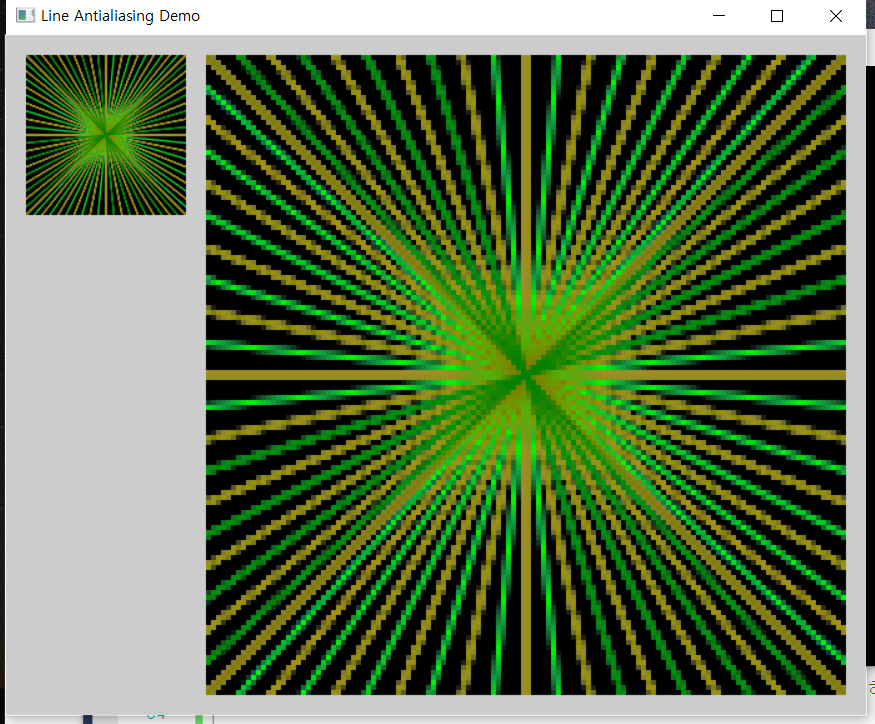
\*보너스 과제 (혼합, 스무딩 함수 사용을 위한 추가적인 옵션)

(1)GL\_FASTEST: 질이 떨어져도 가장 빠르게 그리기 원할시 사용한다

-이렇게 코드를 사용하였습니다.

glHint(GL\_LINE\_SMOOTH\_HINT, GL\_FASTEST);

-실행결과



(2)GL\_DONT\_CARE 는 디폴드 값이며 OpenGL이 알아서 속도와 품질을 결정하게 된다

-이렇게 코드를 사용하습니다.

glHint(GL\_LINE\_SMOOTH\_HINT, GL\_DONT\_CARE);

-실행결과

