2022 정보과학 프로젝트 보고서

20046 박건호

1. 개요

프로세싱 등 많은 것을 해본 사람으로서 2D에서는 명암을 구현하기 너무 어려워 단색으로만 프로그램을 디자인해야한다는 아쉬운 경험이 있었다. 특히 프로그램에서 명암을 넣는다는 것을 고민하는 것보다 사진을 봤을 때 너무 밋밋하여 예쁘게 디자인하고 싶다는 사람도 많을 것이라 생각한다. 프로그램에서 조작하여 이미지를 만든다는 개념보다는 이미지를 실제로 불러와서 조명을 반영한 이미지를 내보내는 프로그램을 만들 것이다.

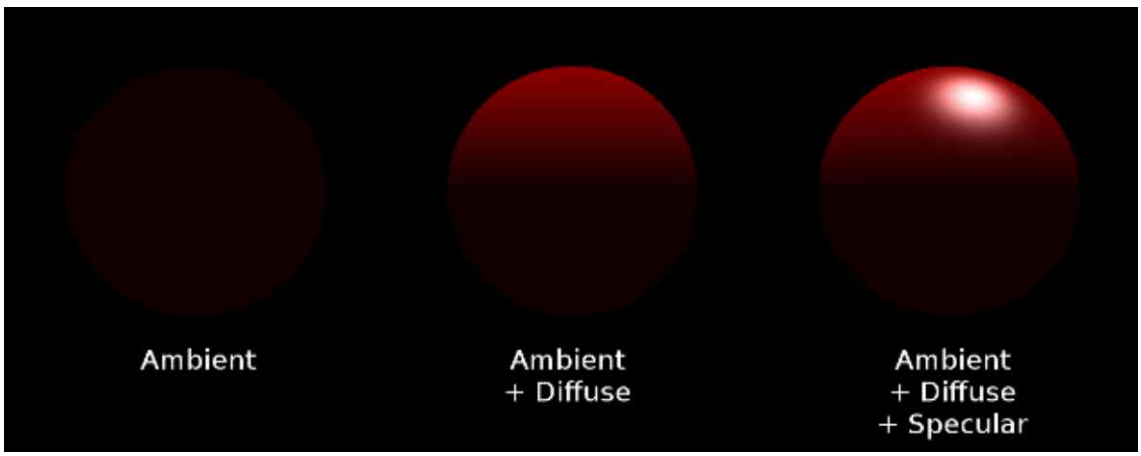
2. 프로젝트의 목표 및 필요성

목표는 위에서 설명했듯이 어떠한 이미지에 조명을 추가하여 그 이미지를 저장하는 것이다. 이 프로젝트를 통해 우리가 원할 때 사진의 조명을 많이 추가 할 수 있을 것이다. 물론 조명을 여러개 추가했을 때 각각의 조명에 대한 특성이 모두 나타나도록 하는 것이 중요한 점이 될 것이다.

3. 프로젝트 수행 과정

3-1 lambert light이란?

어떠한 사진이나 모형에서 명암이 있지 않는다면 단색으로 덮어지는 경향이 있어 3차원 좌표계의 위치를 잘 파악하기 힘들다. 이것을 위해 많이 사용되는 조명 기법인 Lambertian reflectance를 사용하였다. 이는 실재로 mtl 파일에서 사용되는 기법이다.



이 사진에서는 ambient color, diffuse color, specular color의 효과를 보여준다. 여기서 ambient color는 평균적인 색상을 의미하는데 전체적으로 어두워 잘 보이지 않지만 잘보면 평균적인 색상을 나타낸다는 것을 알 수 있다.

다시 정리해보자면
전체적인 평균 색상을 알려주는 ambient 색깔
물체의 조명의 위치에 따라 부분마다 다른 색깔이 나타날 것이다. 이에 대한 효과는 diffuse 색깔로 나타난다.
그리고 마지막으로 그 물체의 특수한 재질때문에 생기는 효과로 인한 색깔이 더해진다.
제일 대표적인 예시로 광택이 있으며 이를 specular 색깔이라고 한다
라고 할 수 있다.

광택의 경우는 재질까지 고려해야하는 부분이기 때문에 하지 않았고 diffuse color와 관련된 내용까지 구현을 해보았다.

1. ambient 색깔만 반영하면 최종 색깔은 다음과 같은 식으로 나타난다.

$$ambientcolor = materialcolor * ambient,$$

2. diffuse 색깔까지 반영하면 최종 색깔은 다음과 같은 식으로 나타난다.

$$lightvector = lightposition - objectposition$$

$$cosine = dotproduct(objectnormalvector(0), normalizedlightvector)$$

$$lambertfactor = max(cosine, 0)$$

$$luminosity = \frac{1}{1 + distance * distance}$$

$$diffusecolor = materialcolor * lightcolor * lambertfactor * luminosity$$

$$finalcolor = ambientcolor + diffusecolor$$

관련 내용 출처: https://raypop.tistory.com/52

3-2 lambert light 기반 Turn Light 구현

```
class TurnLight{
    static int height,width;
    static float[][] Red = new float[500][500];
    static float[][] Green = new float[500][500];
    static float[][] Blue = new float[500][500];
    static float[][] newRed = new float[500][500];
    static float[][] newGreen = new float[500][500];
    static float[][] newBlue = new float[500][500];
    static float lightX,lightY,lightZ;
```

```
    static float lightR,lightG,lightB;
    static float power;
    static float[] Ambient ={(float)0.7,(float)0.7,(float)0.7};
    static Color ambient_color(int x,int y){
        float R,G,B;
        R = Red[x][y]*Ambient[0];
        G = Green[x][y]*Ambient[1];
        B = Blue[x][y]*Ambient[2];
        return Color.rgb((int)R, (int)G, (int)B);
    }
    static Color diffuse_color(int x,int y){
        float R,G,B;
        float[] lightVector = { lightX - x , lightY - y, lightZ };
        float mag =
(float)Math.sqrt((lightX-x)*(lightX-x)+(lightY-y)*(lightY-y)+lightZ*lightZ);
        for(int i=0 ; i<3 ; i++ ){lightVector[i]/=mag;}
        float cosine = lightVector[2];
        float lambertfactor = Math.max(cosine, 0);
        float luminosity = 1/(1+mag*mag);
        R = Math.min(Red[x][y]*power*2*lightR*lambertfactor*luminosity,255);
        G = Math.min(Green[x][y]*power*2*lightG*lambertfactor*luminosity,255);
        B = Math.min(Blue[x][y]*power*2*lightB*lambertfactor*luminosity,255);
        return Color.rgb((int)R, (int)G, (int)B);
    }
    static Color lambert_color(int x,int y){
        Color ambient = ambient_color(x,y);
        Color diffuse = diffuse_color(x,y);
        float finalR,finalG,finalB;
        finalR=(float) Math.min(((ambient.getRed()+diffuse.getRed())*255),255);
        finalG=(float) Math.min(((ambient.getGreen()+diffuse.getGreen())*255),255);
        finalB=(float) Math.min(((ambient.getBlue()+diffuse.getBlue())*255),255);
        return Color.rgb((int)finalR,(int)finalG,(int)finalB);
    }
}
```

3-3 Pixel

이제 이미지 픽셀을 다룰 수 있는 api에 대해서 살펴보아야한다.

이미지의 픽셀을 따오기 위해서 BufferedReader라는 클래스를 사용하였다.

```
BufferedImage buffered = SwingFXUtils.fromFXImage(image, null);
hasAlphaChannel = buffered.getAlphaRaster() != null;
```

이와 같은 코드를 짜면 이미지를 bufferedreader로, 명암부분을 나타내는 alpha가 이 이미지에 사용되었는지 알 수 있다.

그리고 픽셀의 각각 값을 알기 위해서는 PixelReader라는 클래스를 만들어야 한다.
pixelReader = image.getPixelReader();

그리고 pixel 값을 보았더니 가로 세로기준 각각 R,G,B,alpha값을 저장하는데 2차원 배열만을 사용한다.

```
if (hasAlphaChannel)                                    else
pixels[pos++] = (int) color.getBrightness(); // Alpha argb += -16777216; // 255 alpha
pixels[pos++] = (int) (color.getBlue()*255); // Blue
pixels[pos++] = (int) (color.getGreen()*255); // Green
pixels[pos++] = (int) (color.getRed()*255); // Red
```

각각 값이 0부터 255것을 이용하여 2진법 저장 구조를 생각하면 이를 알 수 있다 이 과정을 거치면 나중에 R,G,B,alpha 값을 쉽게 알 수 있다.

3-4 MakeImage

이제 이미지를 만들었다면 이를 저장하는 방법에 대해서 알아보아야 한다.

이것은 WritableImage라는 클래스와 PixelWriter라는 클래스를 사용하면 된다.
이를 각각 선언하면 빈 캔버스마냥 아무것도 없는 상태가 된다. 여기다가 각각의 픽셀을 setColor(x,y,Pixel) 함수를 이용하여 계속 처리해주면 된다.

```
public class MakeImage{
        private static int width,height;
        private static Image making;
        private static PixelWriter write;
        private static WritableImage newimage;
        private static Color[][] resultscolor = new Color[500][500];
        private static int[][] resultsargb = new int[500][500];
        MakeImage(int width,int height){
                this.width=width; this.height=height;
                newimage = new WritableImage(width,height);
                write = newimage.getPixelWriter();
                for(int i=0;i<width;i++){
                        for(int j=0;j<height;j++){
                                MakeImage.setPixel(i,j,Color.rgb(255,255,255));
                        }
```

```
                }
        }
        static void setPixel(int x,int y,Color col){
                write.setColor(x,y,col);
                resultscolor[x][y]=col;
        }
        static void setArgb(int x,int y,int argb){
                write.setArgb(x, y, argb);
                resultsargb[x][y]=argb;
                return;
        }
    static WritableImage returnimage(){
        return newimage;
    }
    static void update(){
        for(int i=0;i<width;i++){
                for(int j=0;j<height;j++){
                        MakeImage.setPixel(i,j,resultscolor[i][j]);
                }
        }
    }
}
```

3-5 RootController

이제 보조하는 클래스에 대해 모두 알아보았으니 입력을 어떻게 받는지 알아보아야 한다.

load: 이미지를 파일탐색기를 이용하여 불러온다

save: 옆에 있는 TextField 의 이름으로 png파일로 조명이 반영된 이미지가 저장된다.

lightlist: 조명을 추가하고 제거할 수 있다.

이 list에서 설정을 바꿀 조명을 선택하고 lightproperty에서 값을 변하게 하면 조명에 대한 정보가 저장된다.

3-6 ChangeImage

```
public class ChangeImage{
    public static int lightX[],lightY[],lightZ[],lightR[],lightG[],lightB[];
    public static float power[],ambient[];   -> 여러 조명들의 정보들을 모두 저장
    public static Image beforeImage ;   -> 조명의 원본
    public static WritableImage temp;   -> 임시 저장 장소
    public static WritableImage newImage; -> 최종 이미지
    public static int now=0; -> 조명이 모두 몇개인가
    ChangeImage(){}
    static void setImage(Image image){
                beforeImage = image;
                TurnLight.width = (int) beforeImage.getWidth();
                TurnLight.height = (int) beforeImage.getHeight();
    } -> 이미지의 높이 너비를 받아온다.
    static boolean saveImage(String base) throws IOException{
        File f = new File(base + ".png");
        return ImageIO.write(SwingFXUtils.fromFXImage(newImage, null),"png", f);
    } -> 최종본인 newImage를 방출
    static void update(){
            ImageInfo Info = new ImageInfo(beforeImage);
              MakeImage t = new
MakeImage((int)beforeImage.getWidth(),(int)beforeImage.getHeight());
              for(int i=0;i<beforeImage.getWidth();i++){
                        for(int j=0;j<beforeImage.getHeight();j++){

MakeImage.setPixel(i,j,Color.rgb(ImageInfo.getRGB(i,j)[0],ImageInfo.getRGB(i,j)[1],ImageI
nfo.getRGB(i,j)[2]));
                        }
                }
            temp=MakeImage.returnimage();
            for(int cnt=1; cnt<=now ; cnt++){
                    Info = new ImageInfo((Image)temp);
                    for(int i=0;i<beforeImage.getWidth();i++){
                        for(int j=0;j<beforeImage.getHeight();j++){
                                TurnLight.Red[i][j]= (float) ImageInfo.getRGB(i,j)[0];
                                TurnLight.Green[i][j]= (float) ImageInfo.getRGB(i,j)[1];
                                TurnLight.Blue[i][j]= (float) ImageInfo.getRGB(i,j)[2];
                        }
                    }
```

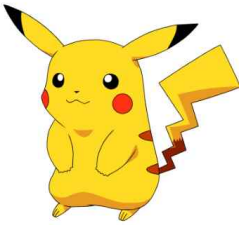
```
TurnLight.lightX=lightX[cnt];TurnLight.lightY=lightY[cnt];TurnLight.lightZ=lightZ[cnt];

TurnLight.lightR=lightR[cnt];TurnLight.lightG=lightG[cnt];TurnLight.lightB=lightB[cnt];
                float[] temparray = new float[3];
                for(int k=0 ;k<3; k++){
                    temparray[k]=ambient[cnt];
                }
                TurnLight.power=power[cnt]; TurnLight.Ambient = temparray;

            for(int i=0;i<beforeImage.getWidth();i++){
                    for(int j=0;j<beforeImage.getHeight();j++){
                            TurnLight.lambert_color(i, j);
                    }
                }

             MakeImage tmp = new
MakeImage((int)beforeImage.getWidth(),(int)beforeImage.getHeight());

             for(int i=0;i<beforeImage.getWidth();i++){
                    for(int j=0;j<beforeImage.getHeight();j++){
                            Color col = TurnLight.lambert_color(i,j);
                            MakeImage.setPixel(i, j, col);
                    }
                }

             temp = MakeImage.returnimage();
               }
               newImage = temp;
    }
}
```

-> 이미지를 temp에 담음 -> temp의 이미지 정보를 ImageInfo를 이용해 받아온다.
-> TurnLight에 정보를 모두 넘김 -> TurnLight에서 모든 정보를 활용해 계산 -> MakeImage를 이용해 그 정보가 담긴 이미지를 생성 -> temp에 저장

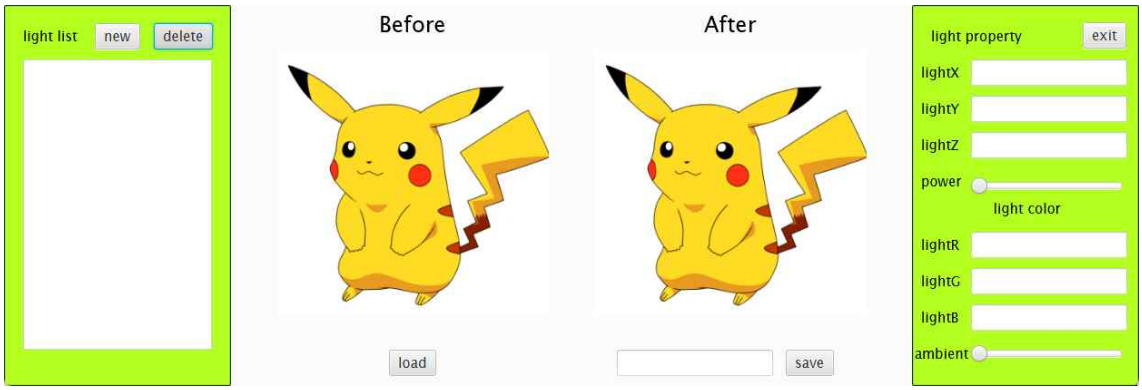
이 과정들을 조명 개수 만큼 반복하여 최종본에 이를 담는 것이 위의 과정이다.

4. 프로젝트 결과

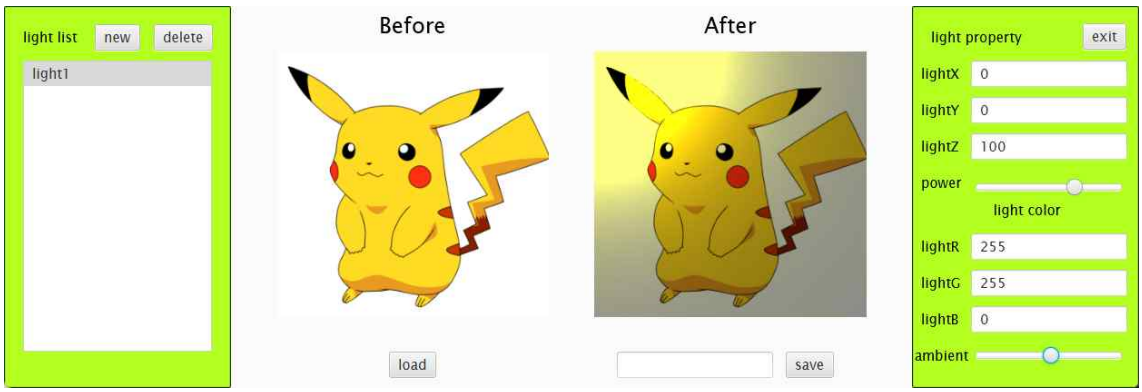
이미지로 피카츄 사진을 가져왔다.



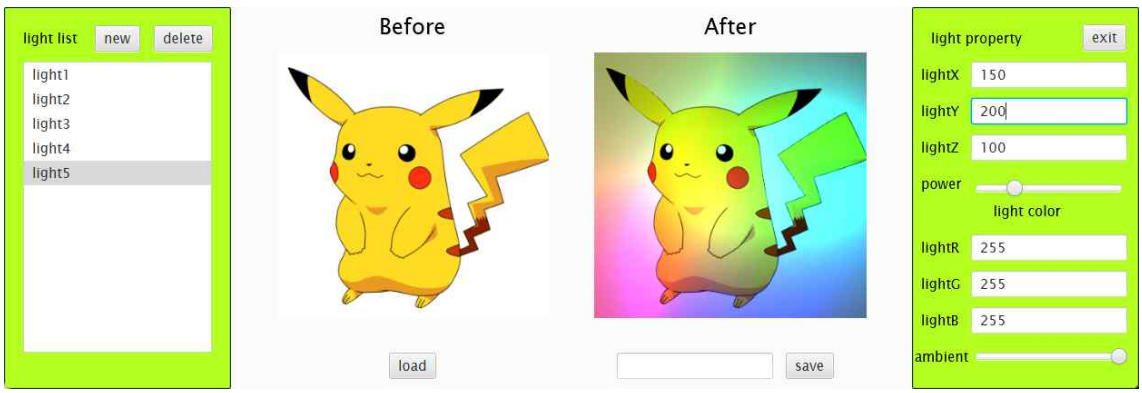
이를 이미지에 가져오면 다음과 같다.

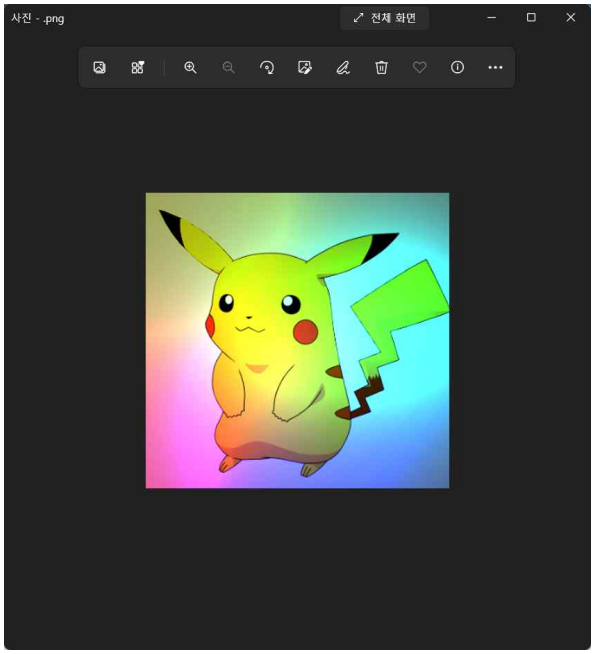


여기서 new를 눌러 조명을 생성하고 property를 잘 만져주면



이런 변화된 화면이 보인다.
이제 조명을 많이 추가한 화면을 보면 다음과 같다
이 사진을 그냥 save를 눌러 저장을 하면 png 사진이 저장된다.





5. 기대효과 및 활용방안

이 과정을 통해 우리는 원하는 수 만큼의 독립된 조명을 설치 할 수 있게 되며 이런 것들은 사진 편집 프로그램이 익숙하지 않을 때 대신 사용할 수 있는 편리함을 제공한다. 그리고 다중 조명을 한번에 놓게 해줌으로서 차별점을 보이기도 한다.

6. 느낀점

처음에는 processing api를 가져와 PImage를 쓸려고 했는데 이렇게 간편한 api가 java에 모두 있다는 점이 놀라웠고 이렇게 1도 모르던 내용을 하나하나 명령어를 찾아보며 프로젝트를 만드니 되게 뿌듯했다. 그리고 이런 과정 속에서 조명이 많이 쓰이는 모델이라서 그런지 실제처럼 너무 잘 나와 기분이 좋았다. 다만 specular light까지 구현 못한점이 아쉽긴 하지만 충분히 만족스러운 프로젝트라 생각했다.