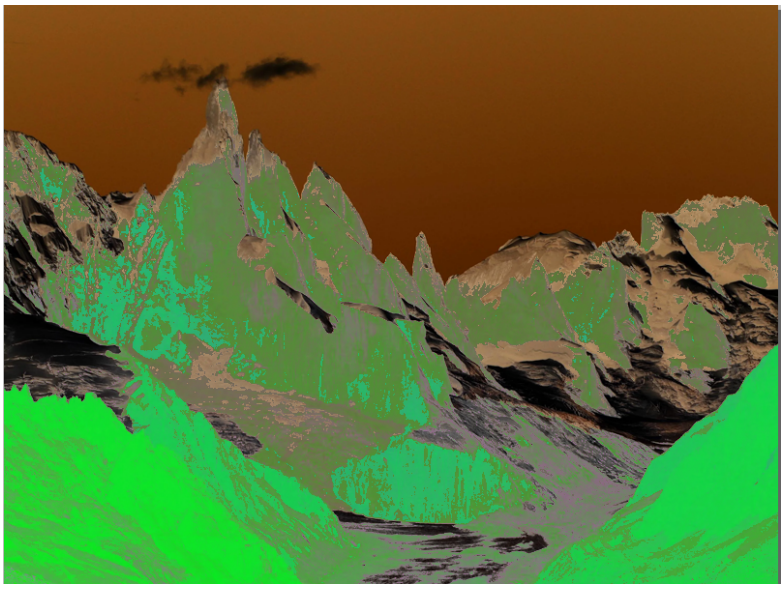
Filters

Link:https://www.openprocessing.org/sketch/1119091

Effect 1-



Code-

```
1    while (idx < img.pixels.length)
2  {
3    color co1 = img.pixels[idx];
4    r = red  (co1);
5    g = green(co1);
6    b = blue (co1);
7      //Create a new color if red pixel color
is over 50 and blue pixel is more than 150
8    if ( r > 50)
9  {
10    if ( b > 150)
11    {
12    color uptColor = color(255-r, 255-g,
255-b);
13    //invert the red pixel to create a new
color
14     }
15     else
16     {
17    //invert the blue color when blue pixel
18     is below 150
19     color uptColor = color(g,255-b,r);
20      }
21  }
22     //Red pixel is below 50
23     else
24   {
25     //Invert the Green Pixel
26     color uptColor = color(r, 255-g, b);
27    }
28
29      img.pixels[idx] = uptColor;
30      idx = idx + 1;
31    }
```

Effect 1 Explanation- In effect 1 it starts with creating a new color if red is over 50 and creating a new color if blue is over 150 in line 8 and 10. Not only that all of the r,g,b colors are inverted as shown in line 12, with all of them subtracted by 255. Now in line 15 we have an "else" where it will invert the blue pixel if it is under 150 and is subtracted by 255. In line 22 we have another "else" where if the red pixel is below 50 it will invert the green pixel and is subtracted by 255.

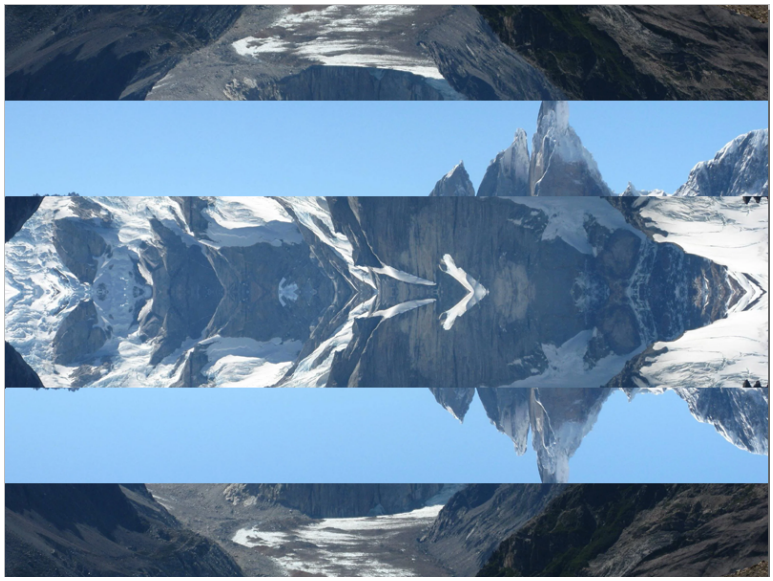
Effect 2-

Code-

```
1    while (index < img.pixels.length)
2 {
3    color co2 = img.pixels[index];
4    r = red  (co2);
5    g = green(co2);
6    b = blue (co2);
7
8     //Update the New color every 10th pixel or
9     every 11th pixel
10     if (index % 10 == 0 || index % 11 == 0)
11  {
12     //Quadruple the red pixel in value and
13     the  Double the green pixel
14     color newColor = color(r*4, g*2, b);
15   }
16     img.pixels[index] = newColor;
17     index = index + 1;
18  }
19
20     img.updatePixels();
21     image(img, imgOffset, imgOffset);
22 }
```

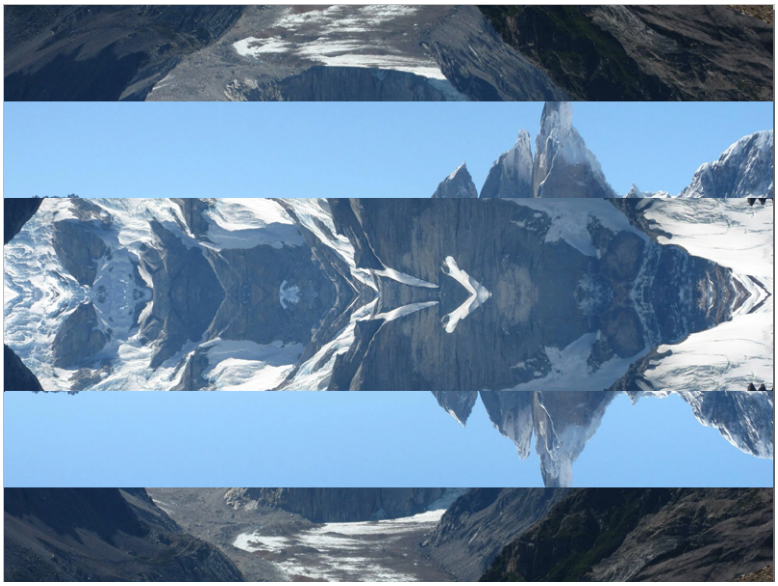
Effect 2 Explanation- In this filter in line 10 it shows that the image will update to a new color every 10th or 11th pixel. In line 14 it shows that the red pixel is multiplied by 4 and the green pixel is multiplied by 2. With all this it is able to make the pixels shown and the sandy yellow orange color.

Effect 3-



Code pt 1-

```
1  while (index < img.pixels.length)
2  {
3    tempPixels[index] = img.pixels[index];
4    index = index + 1;
5   }
6   index = 0;
7   while (index < img.pixels.length)
8    {
9      if (index < img.pixels.length*1/6)
10    {
12      img.pixels[index] =
tempPixels[img.pixels.length-1-index];
13     }
14      if (index > img.pixels.length*2/6 && index  <
img.pixels.length*1/2)
15     {
16 img.pixels[index] =
tempPixels[img.pixels.length-1-index];
17      }
18       if (index > img.pixels.length*4/6 && index <
img.pixels.length*5/6)
19      {
20       img.pixels[index] =
tempPixels[img.pixels.length-1-index];
21       }
22       index = index + 1;
23    }
```



Code pt 2-

```
24     int flippedIndex;
25     index = 0;
26     while (index < img.pixels.length)
27   {
28     int x = index % img.width;
29     int y = int(index/img.width);
30
31     if (index > img.pixels.length*1/6 && index <
32     img.pixels.length*2/6)
33   {
34    flippedIndex = y*img.width + (img.width - 1 -
x);
35     img.pixels[index] =
tempPixels[flippedIndex];
36    }
37     if (index > img.pixels.length*1/2 && index <
38     img.pixels.length*4/6)
39    {
40      flippedIndex = y*img.width + (img.width - 1
41      - x);
42      img.pixels[index] =tempPixels[flippedInde
43    }
44     if (index > img.pixels.length*5/6 && index
< img.pixels.length)
45    {
46     flippedIndex = y*img.width + (img.width - 1
- x);
47     img.pixels[index]=tempPixels[flippedIndex];
48     }
49      index = index + 1;
50     }
```

Effect 3 Explanation- For effect three it is split up into 6 congruent spaces for the image. The first part of the code flips the 1st, 2nd, 5th, and 6th space in the image. It can be seen from lines 9-18 where it shows how it is flipped. In the second part of the code it mirrors the image, so spaces 3 and 4 are mirrored as shown from lines 31-44. The code uses and to help it mirror the image so it can know which spaces to mirror.