Politechnika Świętokrzyska w Kielcach Wydział Informatyki, Elektrotechniki i Automatyki Informatyka, Specjalizacja Grafika Komputerowa		
Grupa: 1ID23B Sebastian Wójcik Bartłomiej Marzec	**Projekt:** Zaawansowane przetwarzanie obrazów	**Data oddania:** 21.03.2022r.
Temat: Greenscreen		

Opis Tematu Projektu:

Tematem naszego projektu było stworzenie aplikacji znanej powszechnie jako Greenscreen. Aplikacja ta polega na wykluczowaniu konkretnego koloru otoczenia, w celu zastąpienia go innym lub dowolnym obrazem w tle. Projekt został wykonany przy wykorzystaniu bibliotek OpenCV oraz dodatkowo OpenGL. Dzięki zastosowainu biblioteki OpenGL i wykorzystaniu shaderów mogliśmy uzyskać zadawalające efekty przy minimalnym wykorzystaniu zasobów komputera. W aplikacji mamy możliwość kluczowania dowolnego koloru otoczenia oraz jego dostosowania w przestrzeni barw HSV. Dodatkowo umożliwiamy opcje dostosowania progu próbkowania barwy tła w celu poprawy już uzyskanych efektów. Obrazy w tle, możemy dowolnie zmieniać. Oprócz samych obrazów, jest również możliwość wyświetlania filmów w postaci najpowszechniej stosowanych rozszerzeń. W celu lepszej organizacji otoczenia, możemy również wydzielć dowolny obszar ekranu w kształcie prostokąta na powieszchni, którego będzie funkcjonować program.

Założenia projektu:

Jako główne założenie projektu postawiliśmy sobie utworzenie prostej aplikacji działającej jak Greenscreen z funkcją próbkowania koloru. Na samym początku chcieliśmy się ograniczyć się do zastosowania jedynie bliblioteki OpenCV, natomiast wszystkie zastosowane przez nas metody implementacji okazały się mało optymalne. Dlatego też zdecydowaliśmy się zaimplementować nasz projekt przy pomocy shaderów, dzięki czemu zwiększyło to diametralnie prędkość aplikacji oraz zmniejszyło obciążenie pamięci komputera. Dodatkowo postanowiliśmy dodać kilka podstawowych funkcjonalności w celu zwiększenia możliwości oprogramowania. Dodaliśmy opcję dowolnej konfiguracji względem poszczególnych komponentów przestrzeni barw HSV. Możemy ustawiać je przy pomocy:

- S – Zwiększenie wartości saturacji
- D – Zmniejszenie wartości saturacji
- V – Zwiększenie jasności barwy
- B – Zmniejszenie jasności barwy

Umożliwiliśmy również dodawanie zarówno obrazów jak i filmów do kolejki, dzięki czemu możemy je dowolnie zmieniać w trakcie działania programu. Zmianę w poszczególnym kierunku wykonujemy za pomocą strzałem w lewo oraz w prawo na klawiaturze. Przez problemy z oświetleniem w trakcie testowania aplikacij doszliśmy do wniosku, iż użyteczną funkcjonalnością byłaby możliwość wyznaczenia obszaru, który podlegałby kluczowaniu. Dlatego też dodaliśmy taką możliwość, którą sterujemy przy pomocy naciśnięcią oraz puszczenia prawego przycisku myszy w dwóch różnych pozycjach.

Wykorzystane biblioteki:

- OpenCV
- Freeglut
- Glew
- Glfw

Dokumentacja Techniczna:

- **GreenscreenOn** – Funkcja włączająca oraz przekazująca argumenty do shadera w celu uruchomienia greenscreena.

```
void GreenScreenOn() {
    shaderUtil.Load("greenscreen.frag");
    shaderUtil.Use();
    glUniform2f(glGetUniformLocation(shaderUtil.mProgramId, "resolution"), sizee.width, sizee.height);
    glUniform1i(glGetUniformLocation(shaderUtil.mProgramId, "tex"), 0);
    glUniform1i(glGetUniformLocation(shaderUtil.mProgramId, "img"), 1);
    glUniform3f(glGetUniformLocation(shaderUtil.mProgramId, "low"), l_colorToDelete[0], l_colorToDelete[1], l_colorToDelete[2]);
    glUniform3f(glGetUniformLocation(shaderUtil.mProgramId, "high"), h_colorToDelete[0], h_colorToDelete[1], h_colorToDelete[2]);
}
```

- **Shader –** Wykonywane są tu wszystki obliczenia związane z podejmowaniem decyzji odnośnie wyświetlanej barwy pixela. Decyzja jest podejmowana na podstawie: pozycji piksela, jego koloru czy przekazanej flagi.

```
#version 130

uniform sampler2D tex;
uniform sampler2D img;
uniform vec3 low;
uniform vec3 high;
uniform int shader;
uniform vec2 start;
uniform vec2 end;
uniform vec2 resolution;

out vec4 color;

float insideBox(vec2 v, vec2 bottomLeft, vec2 topRight) {
    vec2 s = step(bottomLeft, v) - step(topRight, v);
    return s.x * s.y;
}

void main()
{
    vec2 position = (gl_FragCoord.xy/resolution.xy);
    vec4 colorTMP = texture2D(tex,position);
    if(shader == 1){
        if(colorTMP.x > low.x && colorTMP.y > low.y && colorTMP.z > low.z && colorTMP.x < high.x && colorTMP.y < high.y && colorTMP.z < high.z){
            if(start.x == end.x && start.y == end.y){
                color = texture2D(img,position);
            }
            else if(insideBox(gl_FragCoord.xy, start, end)==1) {
                color = texture2D(img,position);
            }else color = colorTMP;
        }else color = colorTMP;
    }else color = colorTMP;
}
```

- **rgb2hsv i hsv2rgb** – Funkcje służące do konwersji przestrzeni barw RGB/HSV.

```
static hsv rgb2hsv(rgb in)
{
    hsv         out;
    double      min, max, delta;

    min = in.r < in.g ? in.r : in.g;
    min = min < in.b ? min : in.b;

    max = in.r > in.g ? in.r : in.g;
    max = max > in.b ? max : in.b;

    out.v = max;
    delta = max - min;
    if (delta < 0.00001)
    {
        out.s = 0;
        out.h = 0;
        return out;
    }
    if (max > 0.0) {
        out.s = (delta / max);
    }
    else {
        out.s = 0.0;
        out.h = NAN;
        return out;
    }
    if (in.r >= max)
        out.h = (in.g - in.b) / delta;
    else
        if (in.g >= max)
            out.h = 2.0 + (in.b - in.r) / delta;
        else
            out.h = 4.0 + (in.r - in.g) / delta;

    out.h *= 60.0;

    if (out.h < 0.0)
        out.h += 360.0;

    return out;
}
```

```
static rgb hsv2rgb(hsv in)
{
    double      hh, p, q, t, ff;
    long        i;
    rgb         out;

    if (in.s <= 0.0) {
        out.r = in.v;
        out.g = in.v;
        out.b = in.v;
        return out;
    }
    hh = in.h;
    if (hh >= 360.0) hh = 0.0;
    hh /= 60.0;
    i = (long)hh;
    ff = hh - i;
    p = in.v * (1.0 - in.s);
    q = in.v * (1.0 - (in.s * ff));
    t = in.v * (1.0 - (in.s * (1.0 - ff)));

    switch (i) { ... }
    return out;
}
```

- **usrednij** – Funkcja ta odpowiada za uśrednienie pobieranej barwy piksela na podstawie jego sąsiadów. Pozwala to na dokładniejsze zidentyfikowanie wartości pobieranego koloru.

```
void usrednij(unsigned char* buff) {
    int R = 0, G = 0, B = 0;
    for (int i = 0; i < 27; i += 3) {
        R += static_cast<int>(buff[i]);
        G += static_cast<int>(buff[i + 1]);
        B += static_cast<int>(buff[i + 2]);
    }
    R /= 9;
    G /= 9;
    B /= 9;

    colorToDelete = Vec3f(R, G, B);
    calculateColor();
    gluniform3f(glGetUniformLocation(shaderUtil.mProgramId, "low"), l_colorToDelete[0], l_colorToDelete[1], l_colorToDelete[2]);
    gluniform3f(glGetUniformLocation(shaderUtil.mProgramId, "high"), h_colorToDelete[0], h_colorToDelete[1], h_colorToDelete[2]);
}
```

- **calculateColor** – Funkcja pozwalająca określić próg próbkowania barwy tła.

```
void calculateColor() {
    Vec3f rangeVector = Vec3f(colorThreshold, colorThreshold, colorThreshold);
    l_colorToDelete = colorToDelete / 255;
    l_colorToDelete -= rangeVector;
    h_colorToDelete = colorToDelete / 255;
    h_colorToDelete += rangeVector;
}
```

Zdjęcia z aplikacji:

Instrukcja sterowania oraz wartość obecnie próbowanego koloru:

```
STEROWANIE:
LPM - Przycisniecie - Probkowanie koloru
PPM - Przycisniecie / Puszczenie - Wybor pozycji prostokata obcinajacego obraz w tle
E - Przycisniecie - Wlaczenie/Wylaczenie Greenscreena
S/D - Przytrzymanie - Zwiekszenie/Zmniejszenie wartosci saturacji
V/B - Przytrzymanie - Zwiekszenie/Zmniejszenie wartosci koloru
+/- - Przytrzymanie - Zwiekszenie/Zmniejszenie progu wybranego koloru
LEWA/PRAWA STRZALKA - Zmiana obrazu w tle
Obecnie kluczowany kolor w przestrzeni HSV oraz prog:
H: 0 S: 0 V: 205 Prog: 0.2Prog: 0.2
```

Kamera z włączonym greenscreenem :
(niedoskonałości związane są z niejednolitym tłem i złym oświetleniem)

Kamera z włączonym greenscreenem i wyznaczonym obszarem kluczowania:

Wnioski i uwagi:

Wszystkie założenia, które ustaliliśmy na początku zostały spełnione. Dodatkowo dodaliśmy kilka funkcji, które początkowo nie były planowane.