Universidad Nacional de San Agustín de Arequipa

Estructuras de Datos Avanzadas

Octree Color Quantization

Integrantes del GRUPO 2A:
- Portocarrero Espirilla, Diego Armando
- Coayla Zuñiga, Gonzalo Eduardo
- Jara Huilca, Arturo Jesús
- Pucho Zevallos, Paul Kelvin

Profesor :
- Machaca Arceda Vicente Enrique

5 de noviembre de 2020

Índice

1. Introducción

El trabajo fue desarrollado en el lenguaje C++ utilizando la libreria de visualización OpenCV

1.1. OpenCV

OpenCV es una biblioteca libre de visión artificial originalmente desarrollada por Intel. OpenCV significa Open Computer Vision (Visión Artificial Abierta). Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en una gran cantidad de aplicaciones, y hasta 2020 se la sigue mencionando como la biblioteca más popular de visión artificial.1 Detección de movimiento, reconocimiento de objetos, reconstrucción 3D a partir de imágenes, son sólo algunos ejemplos de aplicaciones de OpenCV.

1.2. Repositorio

https://github.com/khannom/EDA-Grupo/tree/master/Color%20Quantization

2. Ejercicios

2.1. Definición de Clases

2.1.1. ColorNode

```
struct ColorNode
{
    int level;
    int pixelCount;
    Color color;
    bool isLeaf;
    ColorNode *childs[8];

    ColorNode()
    {
        level = 0;
        pixelCount = 0;
        color = Color(0, 0, 0);
        isLeaf = false;
        for (int i = 0; i < 8; i++)
            childs[i] = nullptr;
    }
    void search_color(cv::Mat &palette);
```

```
    ~ColorNode();

    ColorNode(int level, int pixelCount, Color color, bool isLeaf)
    {
        this->level = level;
        this->pixelCount = pixelCount;
        this->color = color;
        this->isLeaf = isLeaf;

        for (int i = 0; i < 8; i++)
            childs[i] = nullptr;
    }

    void add_level(int height);
    void delete_level();
};

ColorNode::~ColorNode()
{
    if (isLeaf)
        return;

    for (int i = 0; i < 8; i++)
        delete childs[i];
}
```

2.1.2. QuantizationOctree

```
class QuantizationOctree
{
private:
    int levels;
    ColorNode *root;

public:
    QuantizationOctree()
    {
        levels = 7;
        root = new ColorNode(7 - levels, 0, Color(), false);
        root->add_level(levels);
    }

    void fill(cv::Mat &image);
    void reduction();
```

```
    void reconstruction(cv::Mat &image);
    void get_palette(cv::Mat &palette);
    void search_color(ColorNode *path, ColorNode *aux);
    ~QuantizationOctree();
};

QuantizationOctree::~QuantizationOctree()
{
    delete root;
}
```

2.2. Método fill

```
void QuantizationOctree::fill(cv::Mat &image)
{
    int channels = image.channels();

    int nRows = image.rows;
    int nCols = image.cols * channels;

    if (image.isContinuous())
    {
        nCols *= nRows;
        nRows = 1;
    }

    uchar *p;
    ColorNode *path;
    for (int i = 0; i < nRows; ++i)
    {
        p = image.ptr<uchar>(i);
        for (int j = 0; j < nCols; j += 3)
        {
            uchar b = p[j];
            uchar g = p[j + 1];
            uchar r = p[j + 2];
            path = root;

            for (int k = 0; k < levels; k++)
                path = path->childs[getIndex(r, g, b, k)];

            path->color.b += b;
            path->color.g += g;
            path->color.r += r;
```

```
            ++(path->pixelCount);
        }
    }
}
```

2.3. Método Reduction

```
void QuantizationOctree::reduction()
{
    if (!levels)
        return;

    root->delete_level();
    --levels;
}

void ColorNode::delete_level()
{
    if (childs[0]->isLeaf)
    {
        for (int i = 0; i < 8; i++)
        {
            pixelCount += childs[i]->pixelCount;
            color.b += childs[i]->color.b;
            color.g += childs[i]->color.g;
            color.r += childs[i]->color.r;
            delete childs[i];
        }
        isLeaf = true;

        return;
    }

    for (int i = 0; i < 8; i++)
        childs[i]->delete_level();
}
```

2.4. Método Reconstruction

```
void QuantizationOctree::reconstruction(cv::Mat &image)
```

```
{
    int channels = image.channels();

    int nRows = image.rows;
    int nCols = image.cols * channels;

    // Imagen en forma 1 x n
    if (image.isContinuous())
    {
        nCols *= nRows;
        nRows = 1;
    }

    uchar *p;
    ColorNode *path;
    for (int i = 0; i < nRows; ++i)
    {
        p = image.ptr<uchar>(i);
        for (int j = 0; j < nCols; j += 3)
        {
            uchar b = p[j];
            uchar g = p[j + 1];
            uchar r = p[j + 2];
            path = root;

            for (uchar k = 0; k < levels; k++)
                path = path->childs[getIndex(r, g, b, k)];

            p[j] = (path->color.b) / (path->pixelCount);
            p[j + 1] = (path->color.g) / (path->pixelCount);
            p[j + 2] = (path->color.r) / (path->pixelCount);
        }
    }
}
```

2.5. Método Palette

```
void QuantizationOctree::get_palette(cv::Mat &palette)
{
    root->search_color(palette);
}

void ColorNode::search_color(cv::Mat &palette)
{
```

```
    if (isLeaf)
    {
        if (pixelCount > 0)
        {
            uchar colorb = (color.b) / (pixelCount);
            uchar colorg = (color.g) / (pixelCount);
            uchar colorr = (color.r) / (pixelCount);
            palette.push_back(cv::Mat(10, 500, CV_8UC3,
            cv::Scalar(colorb, colorg, colorr)));
        }
        return;
    }

    for (int i = 0; i < 8; i++)
        childs[i]->search_color(palette);
}
```

2.6. Resultados

2.6.1. Main

```
#include "QuantizationOctree.hpp"

#include <opencv2/highgui.hpp>
#include <iostream>

int main(int argc, char *argv[])
{
    cv::Mat image;
    std::string file_name = argv[1];
    image = cv::imread(file_name, CV_LOAD_IMAGE_COLOR);

    if (!image.data)
    {
        std::cout << "Could not open or find the image" << std::endl;
        return -1;
    }

    QuantizationOctree quantization;

    quantization.fill(image);

    quantization.reduction();
    quantization.reduction();
```

```
    quantization.reduction();
    quantization.reduction();
    quantization.reduction();

    cv::Mat palette(0, 500, CV_8UC3, cv::Scalar(0,0,0));

    quantization.get_palette(palette);
    cv::namedWindow("Paleta", cv::WINDOW_AUTOSIZE);
    cv::imshow("Paleta", palette);

    quantization.reconstruction(image);

    cv::namedWindow("Color Quantization", cv::WINDOW_AUTOSIZE);
    cv::imshow("Color Quantization", image);

    cv::Mat original;
    original = cv::imread(file_name, CV_LOAD_IMAGE_COLOR);

    cv::namedWindow("Original", cv::WINDOW_AUTOSIZE);
    cv::imshow("Original", original);

    cv::waitKey(0);

    return 0;
}
```

2.6.2. Makefile

```
make: main.cpp
g++ main.cpp -o main `pkg-config --cflags --libs opencv`
```

2.6.3. Resultados

Foto Original

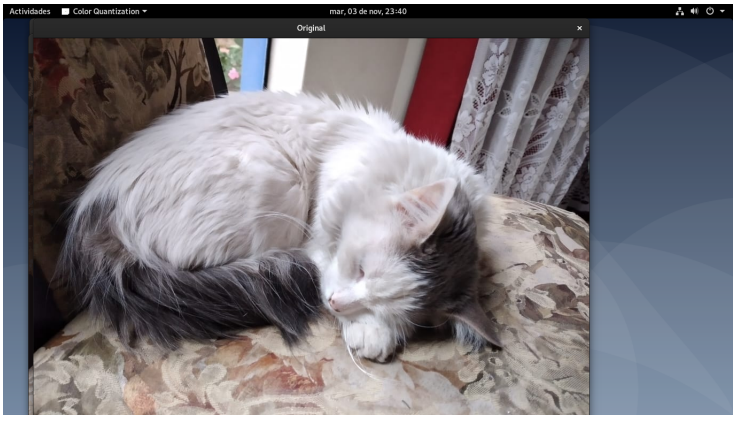
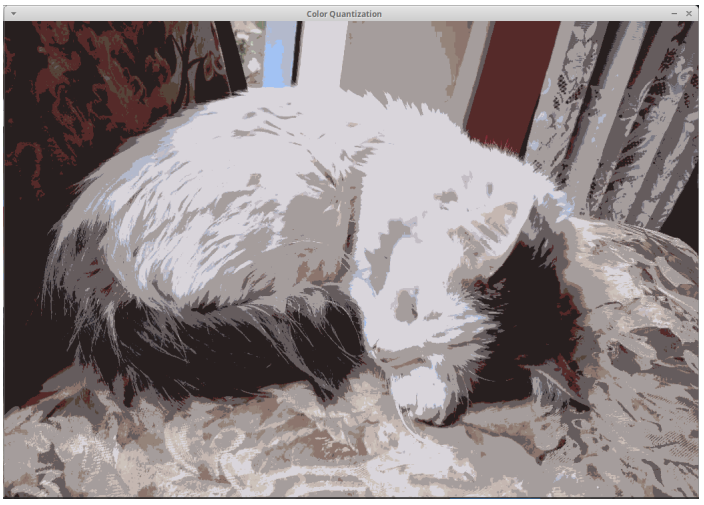


Foto Resultante



Paleta