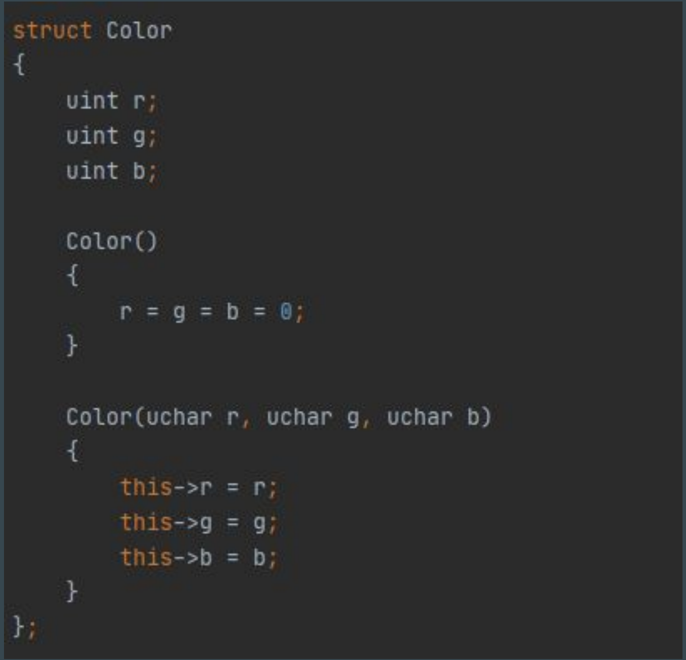
Universidad Nacional de San Agustín de Arequipa
Estructuras de Datos Avanzadas

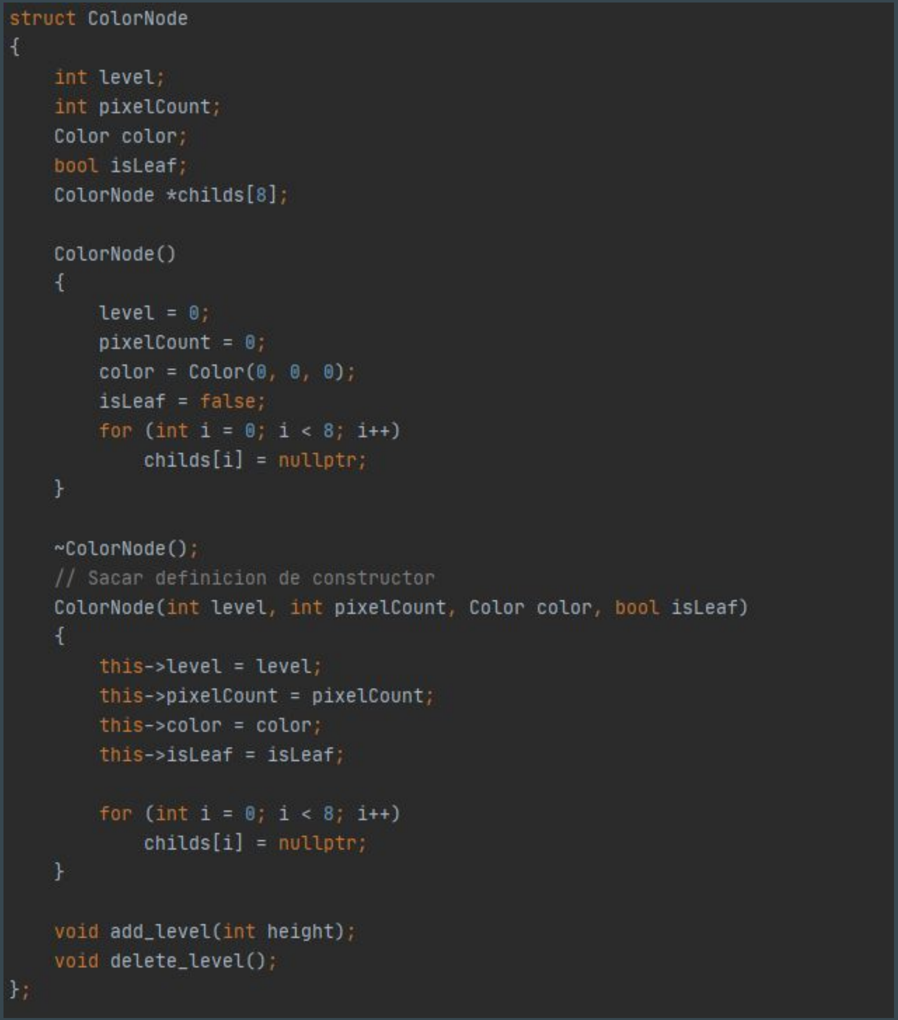
Octree Color Quantization

Integrantes:
Portocarrero Espirilla, Diego Armando
Coayla Zuñiga, Gonzalo Eduardo
Jara Huilca, Arturo Jesús
Pucho Zevallos, Paul Kelvin

Clases Color y ColorNode

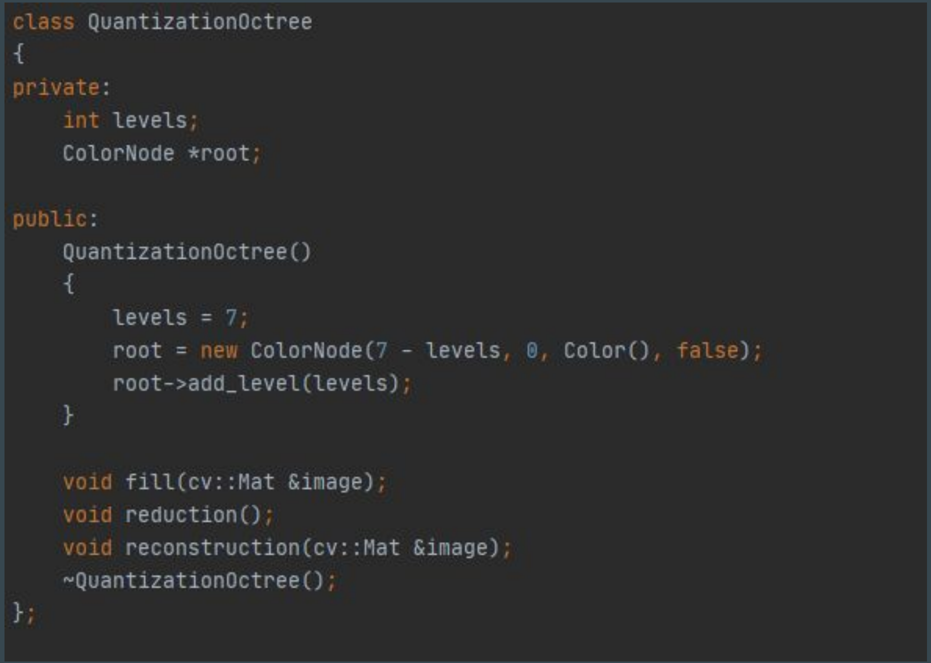


```
struct Color
{
    uint r;
    uint g;
    uint b;

    Color()
    {
        r = g = b = 0;
    }

    Color(uchar r, uchar g, uchar b)
    {
        this->r = r;
        this->g = g;
        this->b = b;
    }
};
```

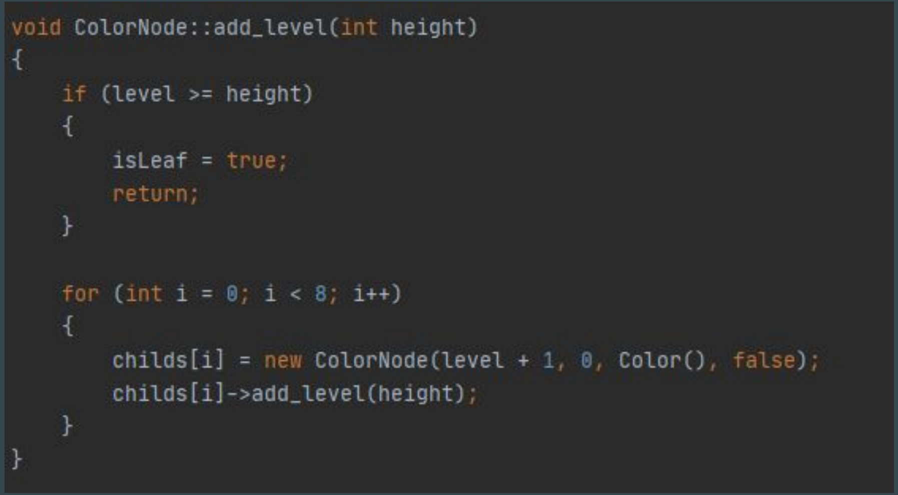


```
struct ColorNode
{
    int level;
    int pixelCount;
    Color color;
    bool isLeaf;
    ColorNode *childs[8];

    ColorNode()
    {
        level = 0;
        pixelCount = 0;
        color = Color(0, 0, 0);
        isLeaf = false;
        for (int i = 0; i < 8; i++)
            childs[i] = nullptr;
    }

    ~ColorNode();
    // Sacar definicion de constructor
    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();
};
```

Clase OcTree e inicialización

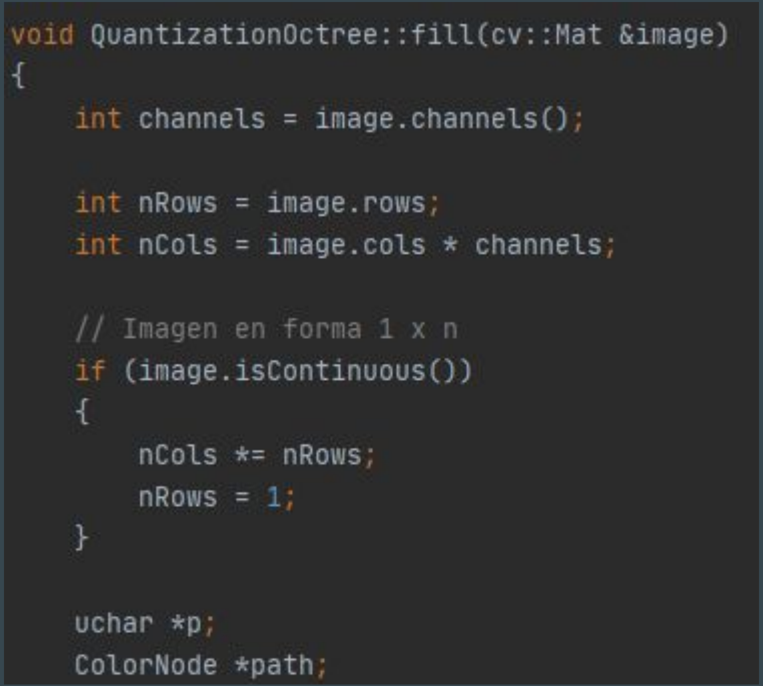


```
class QuantizationOctree
{
private:
    int levels;
    ColorNode *root;

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);
    ~QuantizationOctree();
};
```

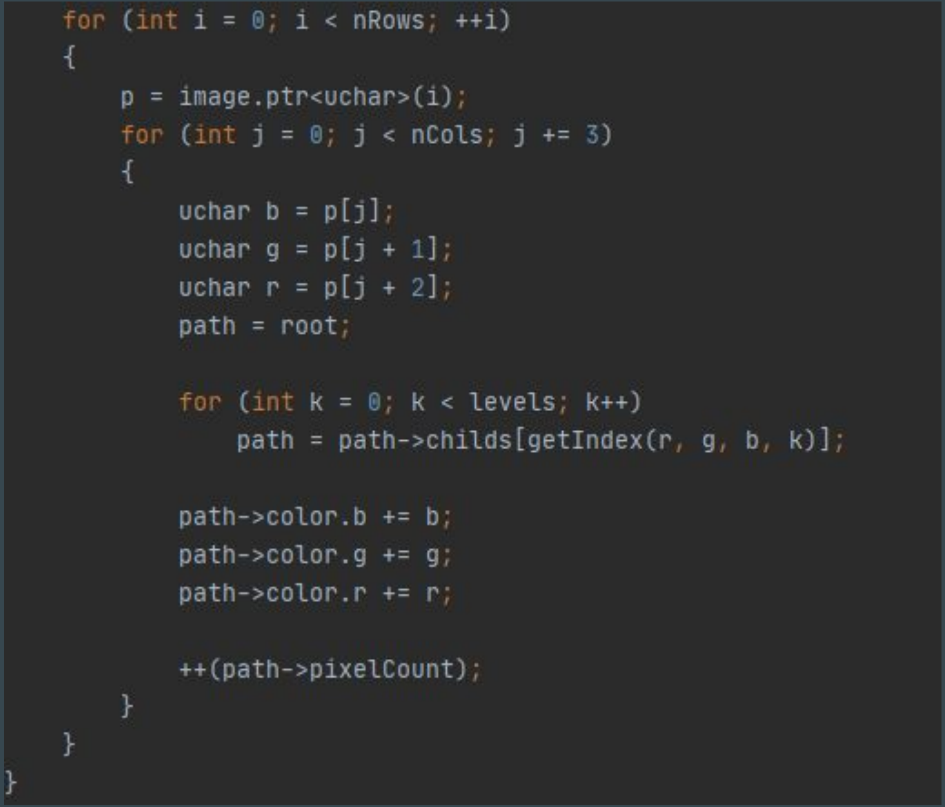


```
void ColorNode::add_level(int height)
{
    if (level >= height)
    {
        isLeaf = true;
        return;
    }

    for (int i = 0; i < 8; i++)
    {
        childs[i] = new ColorNode(level + 1, 0, Color(), false);
        childs[i]->add_level(height);
    }
}
```

Método Fill

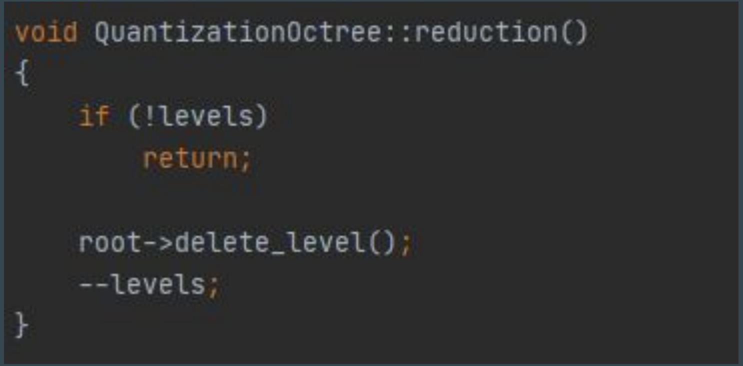


```
void QuantizationOctree::fill(cv::Mat &image)
{
    int channels = image.channels();

    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;
```

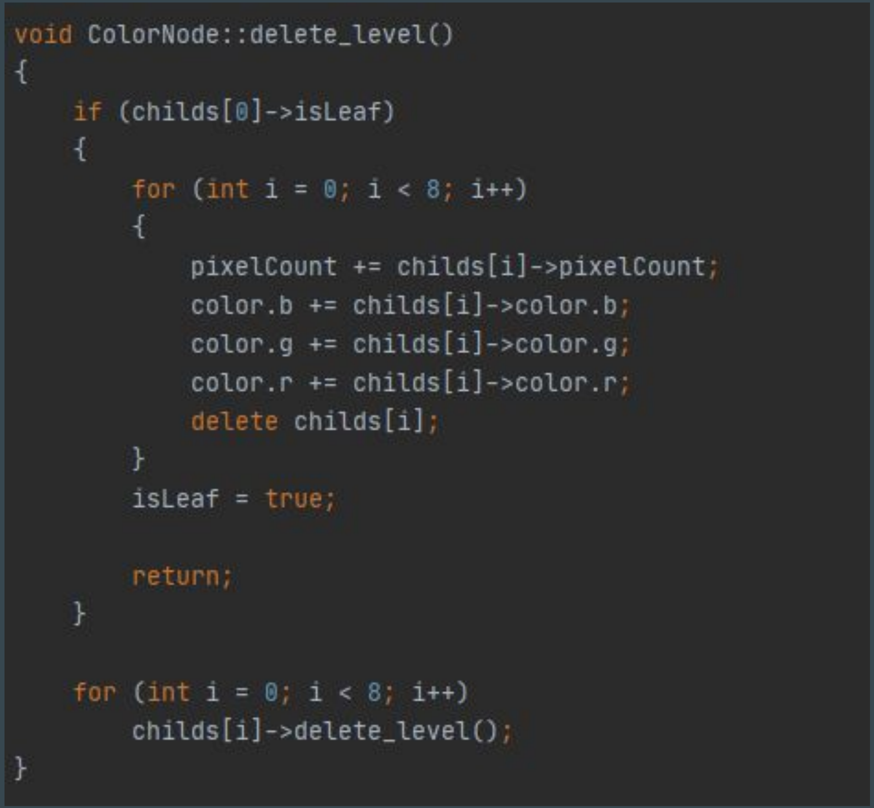


```
    for (int i = 0; i < nRows; ++i)
    {
        p = image.ptr<uchar>(i);
        for (int j = 0; j < nCols; j += 3)
        {
            uchar b = p[j];
            uchar g = p[j + 1];
            uchar r = p[j + 2];
            path = root;

            for (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);
        }
    }
}
```

Método Reducción

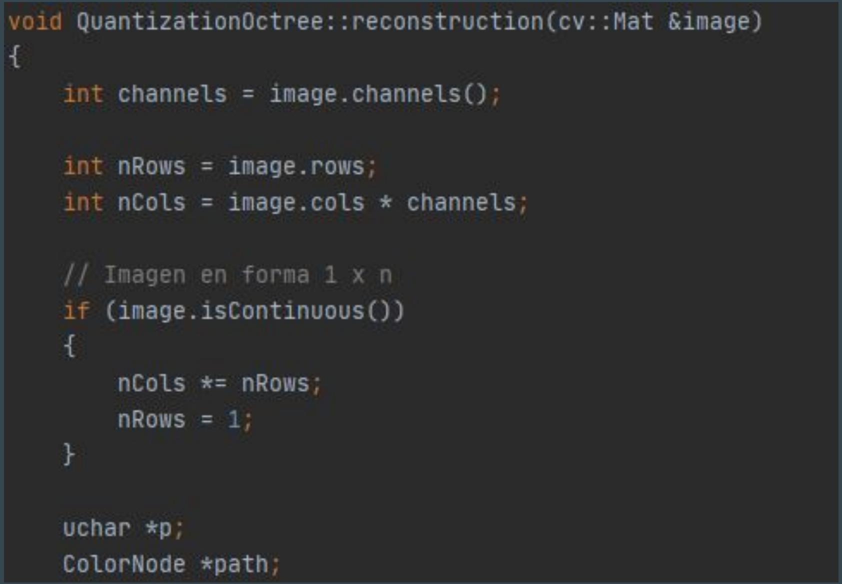


```cpp
void QuantizationOctree::reduction()
{
    if (!levels)
        return;

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

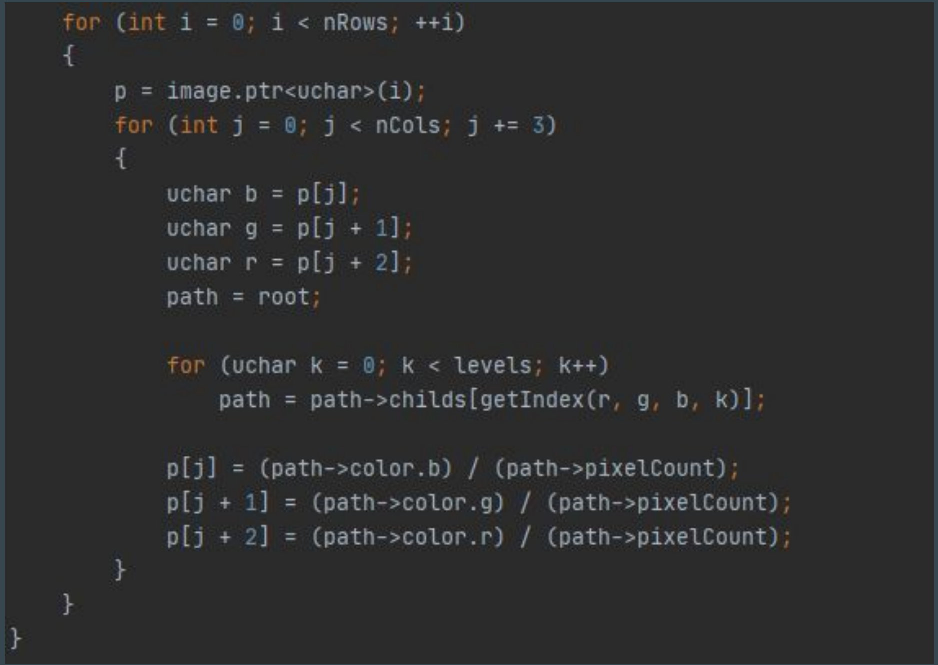


```cpp
void ColorNode::delete_level()
{
    if (childs[0]->isLeaf)
    {
        for (int i = 0; i < 8; i++)
        {
            pixelCount += childs[i]->pixelCount;
            color.b += childs[i]->color.b;
            color.g += childs[i]->color.g;
            color.r += childs[i]->color.r;
   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

        return;
    }

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

Método Reconstrucción

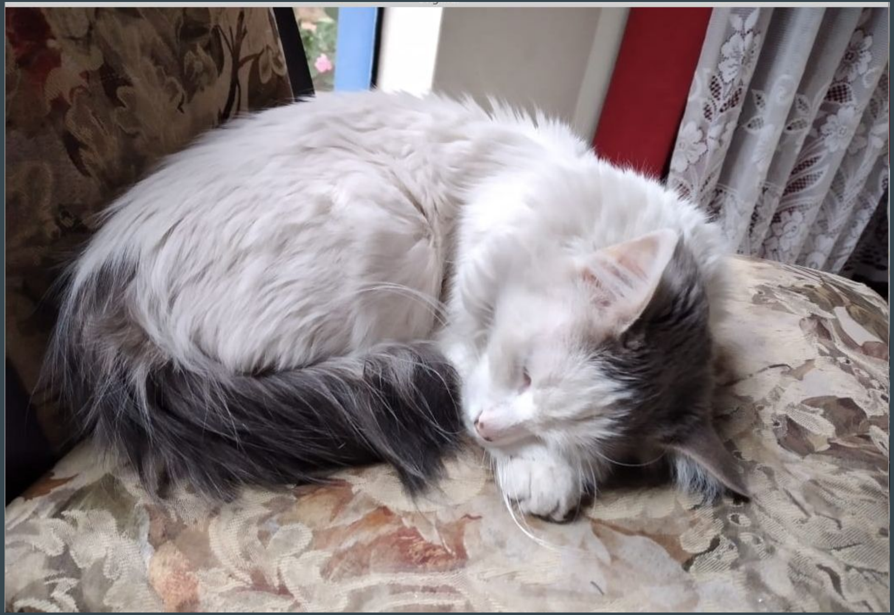


```
void QuantizationOctree::reconstruction(cv::Mat &image)
{
    int channels = image.channels();

    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);
        }
    }
}
```

Imagen original



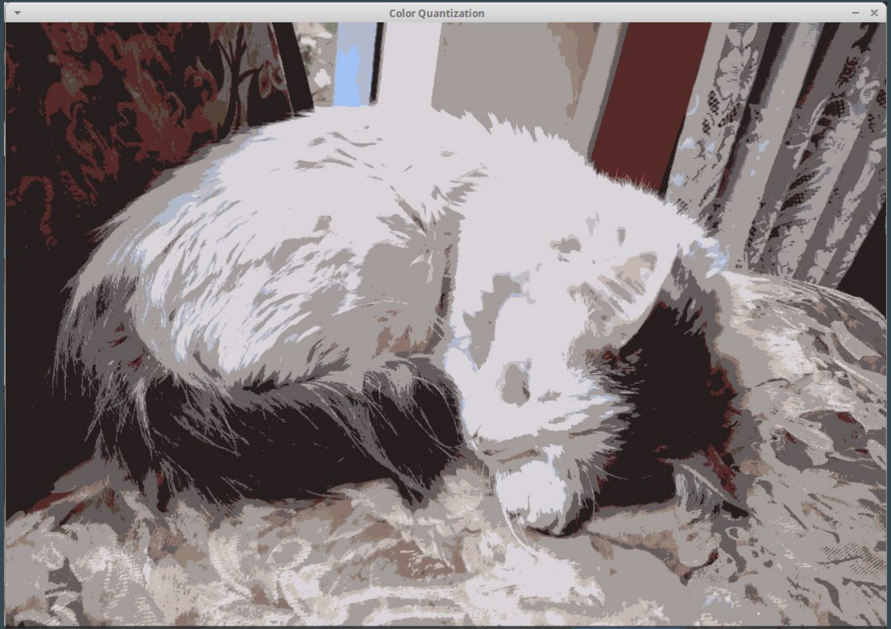
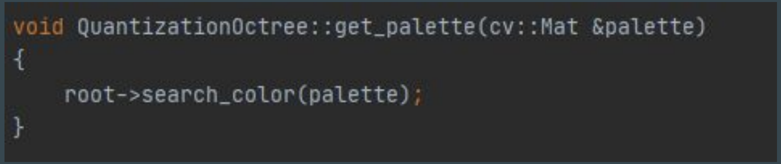
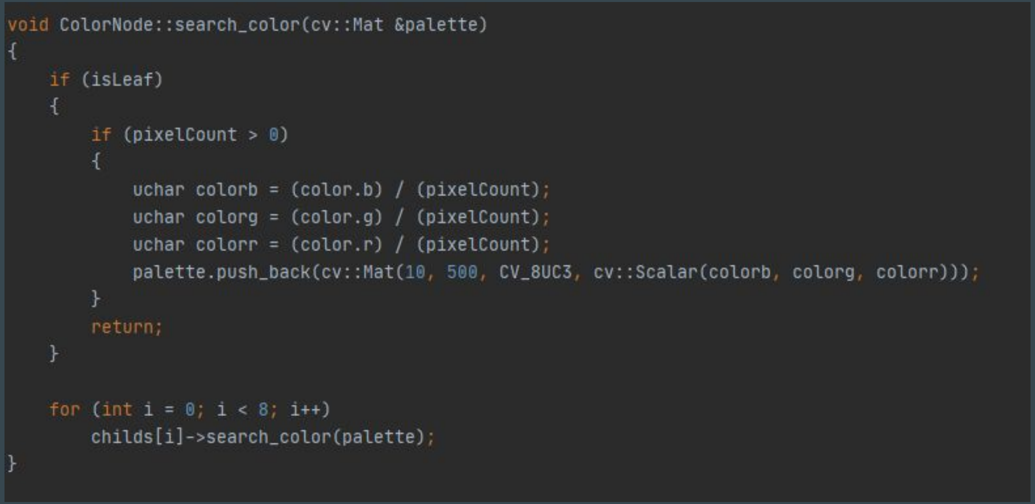


Imagen resultante
(4 reducciones)

Construcción de paleta



```
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);
}
```