using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PPD\_Filter

{

    class PPMOperations

    {

        public PPMImage Image { get; set; }

        public PPMOperations(String file)

        {

            ReadPPMImageFromFile(file);

        }

        public PPMOperations()

        {

        }

        public void ReadPPMImageFromFile(String path)

        {

            List<String> before = File.ReadAllLines(path).ToList();

            //firstLine = beforePicture[0];

            //secondLine = beforePicture[1];

            string[] thirdLine = before[2].Split(' ');

            int width = Convert.ToInt16(thirdLine[0]);

            int height = Convert.ToInt16(thirdLine[1]);

            int maxValue = Convert.ToInt16(before[3].Trim());

            Block[] blocks = RGBListToYUVBlocks(before.Skip(4).ToList(), width, height, 8);

            Image = new PPMImage(before[0], before[1], width, height, maxValue, blocks);

        }

        public void GenerateGrayscaleImage(string path)

        {

            StringBuilder sb = new StringBuilder();

            sb.Append(Image.FirstComment);

            sb.Append('\n');

            sb.Append(Image.SecondComment);

            sb.Append('\n');

            sb.Append(Image.Width + " " + Image.Height);

            sb.Append('\n');

            sb.Append(Image.MaxValue);

            sb.Append('\n');

            sb.Append(YUVBlocksToGrayscaleString());

            File.WriteAllText(path, sb.ToString());

        }

        public void GenerateRegularImage(string path)

        {

            StringBuilder sb = new StringBuilder();

            sb.Append(Image.FirstComment);

            sb.Append('\n');

            sb.Append(Image.SecondComment);

            sb.Append('\n');

            sb.Append(Image.Width + " " + Image.Height);

            sb.Append('\n');

            sb.Append(Image.MaxValue);

            sb.Append('\n');

            sb.Append(YUVBlocksToRGBString());

            File.WriteAllText(path, sb.ToString());

        }

        private static double[,] DivideIntoBlockForIndex(double[,] matrix, int line, int column, int blockSize)

        {

            double[,] blockValues = new double[blockSize, blockSize];

            for (int k = line; k < line + blockSize; k++)

            {

                for (int l = column; l < column + blockSize; l++)

                    blockValues[k - line, l - column] = matrix[k, l];

            }

            return blockValues;

        }

        public static List<Block> DivideIntoBlocks(double[,] yMatrix, double[,] uMatrix, double[,] vMatrix, int blockSize)

        {

            List<Block> blocks = new List<Block>();

            for (int i = 0; i < yMatrix.GetLength(0); i = i + blockSize)

                for (int j = 0; j < yMatrix.GetLength(1); j = j + blockSize)

                {

                    blocks.Add(new Block(Block.Types.Y, blockSize, DivideIntoBlockForIndex(yMatrix, i, j, blockSize), i, j));

                    blocks.Add(new Block(Block.Types.U, blockSize, DivideIntoBlockForIndex(uMatrix, i, j, blockSize), i, j));

                    blocks.Add(new Block(Block.Types.U, blockSize, DivideIntoBlockForIndex(vMatrix, i, j, blockSize), i, j));

                }

            return blocks;

        }

        public static void PopulateYUVMatrices(double[,] yMatrix, double[,] uMatrix, double[,] vMatrix, List<String> values, int height, int width)

        {

            double y, u, v;

            int k = 0;

            for (int i = 0; i < values.Count; i = i + 3)

            {

                y = 0.299 \* Convert.ToInt16(values[i]) + 0.587 \* Convert.ToInt16(values[i + 1]) +

                    0.114 \* Convert.ToInt16(values[i + 2]);

                u = 128 - 0.1687 \* Convert.ToInt16(values[i]) - 0.3312 \* Convert.ToInt16(values[i + 1]) +

                    0.5 \* Convert.ToInt16(values[i + 2]);

                v = 128 + 0.5 \* Convert.ToInt16(values[i]) - 0.4186 \* Convert.ToInt16(values[i + 1]) -

                    0.0813 \* Convert.ToInt16(values[i + 2]);

                yMatrix[k / width, k % width] = y;

                uMatrix[k / width, k % width] = u;

                vMatrix[k / width, k % width] = v;

                k++;

            }

        }

        public List<Block> RGBListToYUVBlocks(List<String> values, int width, int height, int blockDimension)

        {

            List<Block> blocks = new List<Block>();

            double[,] yMatrix = new double[height, width];

            double[,] uMatrix = new double[height, width];

            double[,] vMatrix = new double[height, width];

            PopulateYUVMatrices(yMatrix, uMatrix, vMatrix, values, height, width);

            ////Split the matrices into 8x8 blocks

            //for (int blockLine = 0; blockLine < height / 8; blockLine++)

            //{

            //    for (int blockColumn = 0; blockColumn < width / 8; blockColumn++)

            //    {

            //        double[,] my = new double[8, 8];

            //        double[,] mu = new double[8, 8];

            //        double[,] mv = new double[8, 8];

            //        int mi = 0, mj = 0;

            //        for (int i = blockLine \* 8; i < blockLine \* 8 + 8; i++)

            //        {

            //            mj = 0;

            //            for (int j = blockColumn \* 8; j < blockColumn \* 8 + 8; j++)

            //            {

            //                my[mi, mj] = yMatrix[i, j];

            //                mu[mi, mj] = uMatrix[i, j];

            //                mv[mi, mj] = vMatrix[i, j];

            //                mj++;

            //            }

            //            mi++;

            //        }

            //        Block by = new Block(Block.Types.Y, 8, my, blockLine \* 8, blockColumn \* 8);

            //        Block bu = new Block(Block.Types.U, 8, mu, blockLine \* 8, blockColumn \* 8);

            //        Block bv = new Block(Block.Types.V, 8, mv, blockLine \* 8, blockColumn \* 8);

            //        blocks.Add(by);

            //        blocks.Add(bu);

            //        blocks.Add(bv);

            //    }

            //}

            blocks = DivideIntoBlocks(yMatrix, uMatrix, vMatrix, 8);

            return blocks;

        }

        public int Clamp(double val)

        {

            if (val < 0)

            {

                return 0;

            }

            else if (val > Image.MaxValue)

            {

                return Image.MaxValue;

            }

            else

            {

                return (int)val;

            }

        }

        private Pixel[,] YUVBlocksToPixelMatrix()

        {

            Pixel[,] final = new Pixel[Image.Height, Image.Width];

            for (int i = 0; i < Image.Blocks.Count; i = i + 3)

            {

                double[,] yMatrix = Image.Blocks[i].Matrix;

                double[,] uMatrix = Image.Blocks[i + 1].Matrix;

                double[,] vMatrix = Image.Blocks[i + 2].Matrix;

                int starti = Image.Blocks[i].PositionI;

                int startj = Image.Blocks[i].PositionJ;

                for (int mi = 0; mi < yMatrix.GetLength(0); mi++)

                {

                    for (int mj = 0; mj < yMatrix.GetLength(1); mj++)

                    {

                        double c = yMatrix[mi, mj] - 16;

                        double d = uMatrix[mi, mj] - 128;

                        double e = vMatrix[mi, mj] - 128;

                        int r = Clamp((int)(298 \* c + 409 \* e + 128) >> 8);

                        int g = Clamp((int)(298 \* c - 100 \* d - 208 \* e + 128) >> 8);

                        int b = Clamp((int)(298 \* c + 516 \* d + 128) >> 8);

                        final[starti + mi, startj + mj] = new Pixel(r, g, b);

                    }

                }

            }

            return final;

        }

        private Pixel[,] YUVBlocksToGrayscalePixelMatrix()

        {

            Pixel[,] final = new Pixel[Image.Height, Image.Width];

            for (int i = 0; i < Image.Blocks.Count; i = i + 3)

            {

                double[,] yMatrix = Image.Blocks[i].Matrix;

                int starti = Image.Blocks[i].PositionI;

                int startj = Image.Blocks[i].PositionJ;

                for (int mi = 0; mi < yMatrix.GetLength(0); mi++)

                {

                    for (int mj = 0; mj < yMatrix.GetLength(1); mj++)

                    {

                        double c = yMatrix[mi, mj] - 16;

                        int r = Clamp((int)(298 \* c + 128) >> 8);

                        int g = Clamp((int)(298 \* c + 128) >> 8);

                        int b = Clamp((int)(298 \* c + 128) >> 8);

                        final[starti + mi, startj + mj] = new Pixel(r, g, b);

                    }

                }

            }

            return final;

        }

        public String YUVBlocksToRGBString()

        {

            Pixel[,] final = YUVBlocksToPixelMatrix();

            StringBuilder result = new StringBuilder();

            foreach (Pixel p in final)

            {

                result.Append('\n');

                result.Append(p.R);

                result.Append('\n');

                result.Append(p.G);

                result.Append('\n');

                result.Append(p.B);

            }

            return result.ToString();

        }

        public String YUVBlocksToGrayscaleString()

        {

            Pixel[,] final = YUVBlocksToGrayscalePixelMatrix();

            StringBuilder result = new StringBuilder();

            foreach (Pixel p in final)

            {

                result.Append('\n');

                result.Append(p.R);

                result.Append('\n');

                result.Append(p.G);

                result.Append('\n');

                result.Append(p.B);

            }

            return result.ToString();

        }

    }

}