Computer Vision
Project 2: Noise Filters
C++

Student: Andrew Alleyne

2/22/2021

Algorithm Steps for Noise Filtering

Mean Filter

Step 1: Load the image from the given input text file into an array.

Step 2b Perform mirror framing by dynamically allocating according to frame size. 3x3 frames have a size of 1 and 5x5 2. Mirror it with 1 or 2 layers of pixels by walking the array for the appropriate sides.

Step 3: Perform Average Filtering by taking the neighborhood pixels of the array (either using a template of 3x3 or 5x5) and find the average of each neighboring pixel starting from the first row and column.

Median Filter

Step 1: Load the image from the given input text file into an array.

Step 2: Perform mirror framing by dynamically allocating according to frame size. 3x3 frames have a size of 1 and 5x5 2. Mirror it with 1 or 2 layers of pixels(neighbors) by walking the array for the appropriate sides.

Step 3: Sort the numbers from in ascending order. Bubble sort can be used.

Step 4: Find the median of your pixel's neighborhood including the pixel.

Step 5: Repeat until all pixels are processed.

Corner Preserving Filter

Step 1: Load the image from the given input text file into an array.

Step 2: Obtain neighbors and place them into groups.

Step 3: Compute the difference between the average of the groups.

Step 4: Find the minimum between these differences.

Step 5: Repeat **Step 1 - 4**

Source Code

```
/*
 Andrew Alleyne
CS 381/780 : Computer Vision Project 2
Queens College SP 21

Project 2 (in C++): You are to implement the three image enhancement methods
taught in class: (1) 3X3 averaging, (2) 3x3 median filter,(3)5x5 corner
preserving filter.

You can use: "./a.out inFile.txt 30 outfile{1..10}.txt" to create and run this program after
compiling the code.

 */

using namespace std;
#include <iostream>
#include <fstream>
#include <string>
#include <cmath>
#include <climits>

class Img
{

private:
    int numRows;
    int numCols;
    int minVal;
    int maxVal;
    int newMin; //minVal
    int newMax; //maxVal

    //int **mirror5by5Ary;
    int neighborAry[9];
    int neighbor5x5[5][5];
    int neighbor5x5CP[5][5];
    int trackerCP[8];
    double avgCP[8];

    int **medianAry;
    int **mirror5by5AryCP;
    int **mirror3by3Ary;
    int **mirror5by5Ary;
    int **avgAry;
    int thrVal;
    ofstream ofs1;
    ofstream ofs2;
    ifstream ifs;
    bool initializeMinMax = true;
    bool initializeMedMinMax = true;
    bool initializeCPMinMax = true;
```

```
    int CPmask[8][5][5] = {
        //  G1 MASK
        {{0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0},
         {0, 0, 1, 0, 0},
         {0, 1, 1, 1, 0},
         {1, 1, 1, 1, 1}},
        //  G1 MASK
        {{1, 0, 0, 0, 0},
         {1, 1, 0, 0, 0},
         {1, 1, 1, 0, 0},
         {1, 1, 0, 0, 0},
         {1, 0, 0, 0, 0}},
        //  G3 MASK
        {{1, 1, 1, 1, 1},
         {0, 1, 1, 1, 0},
         {0, 0, 1, 0, 0},
         {0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0}},
        //  G4 MASK
        {{0, 0, 0, 0, 1},
         {0, 0, 0, 1, 1},
         {0, 0, 1, 1, 1},
         {0, 0, 0, 1, 1},
         {0, 0, 0, 0, 1}},
        //  G5 MASK
        {{1, 1, 1, 0, 0},
         {1, 1, 1, 0, 0},
         {1, 1, 1, 0, 0},
         {0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0}},
        //  G6 MASK
        {{0, 0, 1, 1, 1},
         {0, 0, 1, 1, 1},
         {0, 0, 1, 1, 1},
         {0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0}},
        //  G7 MASK
        {{0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0},
         {0, 0, 1, 1, 1},
         {0, 0, 1, 1, 1},
         {0, 0, 1, 1, 1}},
        //  G8 MASK
        {{0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0},
         {1, 1, 1, 0, 0},
         {1, 1, 1, 0, 0},
         {1, 1, 1, 0, 0}}

    };

public:
    Img(ifstream &ifs, ofstream &ofs2, int thrVal)
    {

        this->thrVal = thrVal;

        if (ifs)
        {
            ifs >> this->numRows >> this->numCols >> this->minVal >> this->maxVal;
        }
```

```
    mirror3by3Ary = new int *[numRows + 2];
    medianAry = new int *[numRows + 2];
    avgAry = new int *[numRows + 2];

    for (int i = 0; i <= numRows + 2; i++)
    {
        mirror3by3Ary[i] = new int[numCols + 2];
        medianAry[i] = new int[numRows + 2];
        avgAry[i] = new int[numCols + 2];
    }

    mirror5by5Ary = new int *[numRows + 4];
    mirror5by5AryCP = new int *[numRows + 4];

    for (int i = 0; i <= numRows + 4; i++) //Seg fault happened here
    {

        mirror5by5Ary[i] = new int[numCols + 4];
        mirror5by5AryCP[i] = new int[numCols + 4];
    }
}

void loadImage(ifstream &ifs)
{
    int pixel;

    for (int i = 1; i <= numRows; i++)
    {

        for (int j = 1; j <= numCols; j++)
        {
            if (ifs >> pixel)
            {
                mirror3by3Ary[i][j] = pixel;
                mirror5by5Ary[i][j] = pixel;
            }
        }
    }
}

//Should be able to handle 3x3 and 5x5;
void
mirrorFraming(int frameSize)
{
    /*
     padded already with 0's
     todo
    0 0 0 0 0
    0 1 2 3 0
    0 4 5 6 0
    0 7 8 9 0
    0 0 0 0 0


   now mirror the grayscale values - Mirror Framing*/
    if (frameSize == 1)
    {
        for (int i = 0; i <= numRows + 1; i++)
        {
            mirror3by3Ary[i][0] = mirror3by3Ary[i][1];                  //left
            mirror3by3Ary[i][numCols + 1] = mirror3by3Ary[i][numCols]; //right
        }

        for (int i = 0; i <= numCols + 1; i++)
```

```
            {
                mirror3by3Ary[0][i] = mirror3by3Ary[1][i];                 //top
                mirror3by3Ary[numRows + 1][i] = mirror3by3Ary[numRows][i]; //bot
            }
        }
        else if (frameSize == 2)
        {
            for (int i = 0; i <= numRows + 4; i++)
            {
                mirror5by5Ary[i][0] = mirror5by5Ary[i][3]; //left
                mirror5by5Ary[i][1] = mirror5by5Ary[i][2]; //left

                mirror5by5Ary[i][numCols + 1] = mirror5by5Ary[i][numCols];     //right
                mirror5by5Ary[i][numCols + 2] = mirror5by5Ary[i][numCols - 1]; //right
            }

            for (int i = 0; i <= numCols + 4; i++)
            {
                mirror5by5Ary[0][i] = mirror5by5Ary[3][i]; //top
                mirror5by5Ary[1][i] = mirror5by5Ary[2][i]; //top

                mirror5by5Ary[numRows + 1][i] = mirror5by5Ary[numRows][i];     //bot
                mirror5by5Ary[numRows + 2][i] = mirror5by5Ary[numRows - 2][i]; //bot
            }
        }
    }

    void computeAverage(ofstream &ofs2, ofstream &ofs3, ofstream &ofs4)
    {
        //Need neighbors first
        for (int r = 1; r <= numRows; r++)
        {
            for (int c = 1; c <= numCols; c++)
            {
                neighborhood(r, c);

                // 3X3 = 9 elements
                //    1 1 2 3 3
                //    1 1 2 3 3
                //    4 4 5 6 6  ==>  1 + 1 + 2 + 1 + 1 + 1 + 2 + 4 + 4 + 5 / 9 = 2.4 ==>
ceil(2.4) ==> 3.0
                //    7 7 8 9 9
                //    7 7 8 9 9
                // Add all neighbors and take the ceiling of them.

                double outputX = 0;
                for (int i = 0; i < 9; i++)
                {

                    outputX = outputX + neighborAry[i];
                }

                double outputX_Avgs = outputX / 9.0;
                avgAry[r][c] = ceil(outputX_Avgs);

                int ceil_ = avgAry[r][c];

                /*
                The new min  and macx should be the first value of average
                newMin = 2;
                newMax = 2;
                */

                setMin_MaxAvg(ceil_);
```

```
        }
    }
    imgReformat2(avgAry, ofs2, 1);
    threshold(thrVal, ofs3, ofs4, avgAry);
}

void neighborhood(int r, int c)
{
    int valAtIndex = 0;
    int neigh = 0;
    for (int i = r - 1; i <= r + 1; i++)
    {
        for (int j = c - 1; j <= c + 1; j++)
        {
            neighborAry[valAtIndex] = mirror3by3Ary[i][j];
            valAtIndex++;
        }
    }
}

void setMin_MaxAvg(int ceil_)
{

    if (initializeMinMax)
    {
        newMin = avgAry[0][0];
        newMax = avgAry[0][0];
        initializeMinMax = false;
    }

    newMin = min(newMin, ceil_);
    newMax = max(newMax, ceil_);
}

//  Revise algorithm on the spec. imgRef print the avg3X3 from r = 1 and c = 1
// imgReformat (avgAry, AvgOutImg, 1)
void imgReformat2(int **arr, ofstream &ofs2, int frameSize)
{
    ofs2 << numRows << " " << numCols << " " << newMin << " " << newMax << endl;

    if (frameSize = 1)
    {
        for (int i = 1; i < numRows + frameSize; i++)
        {
            for (int j = 1; j < numCols + frameSize; j++)
            {
                ofs2 << arr[i][j] << " ";
            }

            ofs2 << endl;
        }
    }
    else if (frameSize = 2)
    {
        for (int i = 0; i < numRows + frameSize; i++)
        {
            for (int j = 0 - 1; j < numCols + frameSize; j++)
            {
                ofs2 << arr[i][j] << " ";
            }

            ofs2 << endl;
        }
    }
```

```
    //Applying thresholding
}

void imgReformat3(int **arr, ofstream &ofs2, int frameSize)
{
    for (int i = 0; i <= numRows + frameSize; i++)
    {
        for (int j = 0; j <= numCols + frameSize; j++)
        {
            ofs2 << arr[i][j] << " ";
        }

        ofs2 << endl;
    }
}

void threshold(int thrVal, ofstream &ofs3, ofstream &ofs4, int **arry)
{
    //  threshold (ary1, ary2, frameSize)

    for (int i = 0; i <= numRows; i++)
    {
        for (int j = 0; j <= numCols; j++)
        {
            if (arry[i][j] >= thrVal)
            {

                ofs3 << 1 << " ";
                ofs4 << 1 << " ";
            }

            if (arry[i][j] < thrVal)
            {
                ofs3 << 0 << " ";
                ofs4 << "."
                     << " ";
            }
        }

        ofs3 << endl;
        ofs4 << endl;
    }
}

// imgReformat (inAry, ostream)
void imgReformat(ofstream &rfImg)
{
    for (int i = 0; i <= numRows + 1; i++)
    {
        for (int j = 0; j <= numCols + 1; j++)
        {
            rfImg << mirror3by3Ary[i][j] << " ";
        }

        rfImg << endl;
    }
}

//     computeMedian(...) // see algorithm below.
// // Scans thru all pixels inside the frame of the mirror3by3Ary,
// // apply median3x3 on each pixel, then, outputs the result to
// // medianAry */
```

```
void computeMedian(ofstream &ofs4, ofstream &ofs5, ofstream &ofs6)
{
    //Need neighbors first
    for (int r = 1; r <= numRows; r++)
    {
        for (int c = 1; c <= numCols; c++)
        {
            neighborhood(r, c);

            // 3X3 = 9 elements
            //   | 1 1 2 |3 3
            //   | 1 1 2 |3 3
            //   |4 4 5 |6 6  ==> BubbleSort
            //   -------
            //    7 7 8 9 9
            //    7 7 8 9 9
            // Sort the neighbors and take the ceiling of them.

            int medianX;

            //Sort items. Need more optimized algo  for sorting bigger array
            bubbleSort(neighborAry);
            int medianAryX;
            int size = sizeof(neighborAry) / sizeof(neighborAry[0]);

            //walk neigh array
            for (int i = 0; i < size; i++)
            {

                if (i == 4)
                {
                    medianAryX = neighborAry[i];
                }
            }

            medianAry[r][c] = medianAryX;

            int median = medianAry[r][c];

            setMin_MaxMedian(median);
        }
    }
    imgReformat2(medianAry, ofs4, 1);
    threshold(thrVal, ofs5, ofs6, medianAry);
}

void bubbleSort(int *neighborAry)
{

    int i, j;
    bool swapped;
    for (i = 0; i < 9 - 1; i++)
    {
        swapped = false;
        for (j = 0; j < 9 - i - 1; j++)
        {
            if (neighborAry[j] > neighborAry[j + 1])
            {
                swap(&neighborAry[j], &neighborAry[j + 1]);
                swapped = true;
            }
        }
    }
}
```

```
    void swap(int *xp, int *yp)
    {
        int temp = *xp;
        *xp = *yp;
        *yp = temp;
    }

    void setMin_MaxMedian(int median)
    {

        if (initializeMedMinMax)
        {
            newMin = avgAry[0][0];
            newMax = avgAry[0][0];
            initializeMedMinMax = false;
        }

        newMin = min(newMin, median);
        newMax = max(newMax, median);
    }

    void computeCPfilter(int frameSize, ofstream &ofs8, ofstream &ofs9, ofstream &ofs10)
    {

        //Need neighbors first
        for (int r = 2; r <= numRows + 2; r++)
        {
            for (int c = 2; c <= numCols + 2; c++)
            {
                double sum = 0;
                int tracker = 0;
                int diff = INT_MAX;

                loadneighbors(r, c);

                for (int pages = 0; pages < 8; pages++)
                {

                    double total = 0;
                    for (int cpmaskROW = 0; cpmaskROW < 5; cpmaskROW++)
                    {

                        for (int cpmaskCOL = 0; cpmaskCOL < 5; cpmaskCOL++)
                        {
                            if (CPmask[pages][cpmaskROW][cpmaskCOL] <= 0)
                            {
                                neighbor5x5CP[cpmaskROW][cpmaskCOL] = 0;
                            }

                            if (CPmask[pages][cpmaskROW][cpmaskCOL] >= 1)
                            {
                                neighbor5x5CP[cpmaskROW][cpmaskCOL] =
neighbor5x5[cpmaskROW][cpmaskCOL];
                            }
                        }
                    }

                    for (int j = 0; j < 5; j++)
                    {
                        for (int k = 0; k < 5; k++)
                        {

                            sum += neighbor5x5CP[j][k];
                        }
```

```
                    }
                    total = sum / 8;

                    avgCP[pages] = total;
                }
                for (int i = 0; i < 8; i++)
                {
                    trackerCP[i] = avgCP[i] -= neighbor5x5[r][c];
                }

                for (int i = 0; i < 8 - 1; i++)
                {
                    for (int j = i + 1; j < 8; j++)
                    {

                        if (abs(trackerCP[i] - trackerCP[j]) < diff)
                        {
                            diff = abs(trackerCP[i] - trackerCP[j]);
                        }
                    }
                }
                setMin_MaxCP(abs(diff));
                mirror5by5AryCP[r][c] = diff;
            }
        }

        //Need to keep track of newMin and newMax
        ofs9 << numRows << " " << numCols << " " << newMin << " " << 1 << endl;
        ofs10 << numRows << " " << numCols << " " << newMin << " " << 1 << endl;

        imgReformat2(mirror5by5Ary, ofs8, frameSize);
        threshold(thrVal, ofs9, ofs10, mirror5by5AryCP);
    }

    void setMin_MaxCP(int ceil_)
    {

        if (initializeCPMinMax)
        {
            newMin = neighbor5x5[0][0];
            newMax = neighbor5x5[0][0];
            initializeCPMinMax = false;
        }

        newMin = min(newMin, ceil_);
        newMax = max(newMax, ceil_);
    }

    void loadneighbors(int r, int c)
    {
        int valAtIndex1 = 0;
        for (int i = r - 2; i <= r + 2; i++)
        {
            int valAtIndex2 = 0;
            for (int j = c - 2; j <= c + 2; j++)
            {
                neighbor5x5[valAtIndex1][valAtIndex2] = mirror5by5Ary[i][j];
                valAtIndex2++;
            }
            valAtIndex1++;
        }
    }
};
```

```
int main(int argc, char *argv[])
{

    int frameSize3x3 = 1;
    int frameSize5x5 = 2;

    //Input and output files
    if (argc < 3)
    {
        cout << " Missing arguments. Your arguments should look like  \" ./a.out, argv[1] -->
[inFile.txt], argv[2] --> [Threshold Value], argv[3 - 12] --> outFiles \"" << endl;
    }

    //Open output and input files.
    ifstream ifs;
    ifs.open(argv[1]);

    ofstream ofs[12];

    ofstream ofs1;
    ofs1.open(argv[3]); //rfImg

    ofstream ofs2;
    ofs2.open(argv[4]); //avgOutImg

    ofstream ofs3;
    ofs3.open(argv[5]); //AvgThrImg

    ofstream ofs4;
    ofs4.open(argv[6]); //AvgPrettyPrint

    ofstream ofs5;
    ofs5.open(argv[7]); //MedianOurImg

    ofstream ofs6;
    ofs6.open(argv[8]);

    ofstream ofs7;
    ofs7.open(argv[9]);

    ofstream ofs8;
    ofs8.open(argv[10]);

    ofstream ofs9;
    ofs9.open(argv[11]);

    ofstream ofs10;
    ofs10.open(argv[12]);

    //Threshold values.
    int Threshold = stoi(argv[2]);

    //Image Class.
    Img
        image(ifs, ofs2, Threshold);

    //Image methods
    image.loadImage(ifs);
    image.imgReformat(ofs1);

    image.mirrorFraming(frameSize3x3);
    image.computeAverage(ofs2, ofs3, ofs4);
    image.computeMedian(ofs5, ofs6, ofs7);
```

```
    image.mirrorFraming(frameSize5x5);
    image.computeCPfilter(frameSize5x5, ofs8, ofs9, ofs10);

    //Close all streams
    ifs.close();
    for (int i = 0; i < 12; i++)
    {
        ofs[i].close();
    }
}
```

Program Output

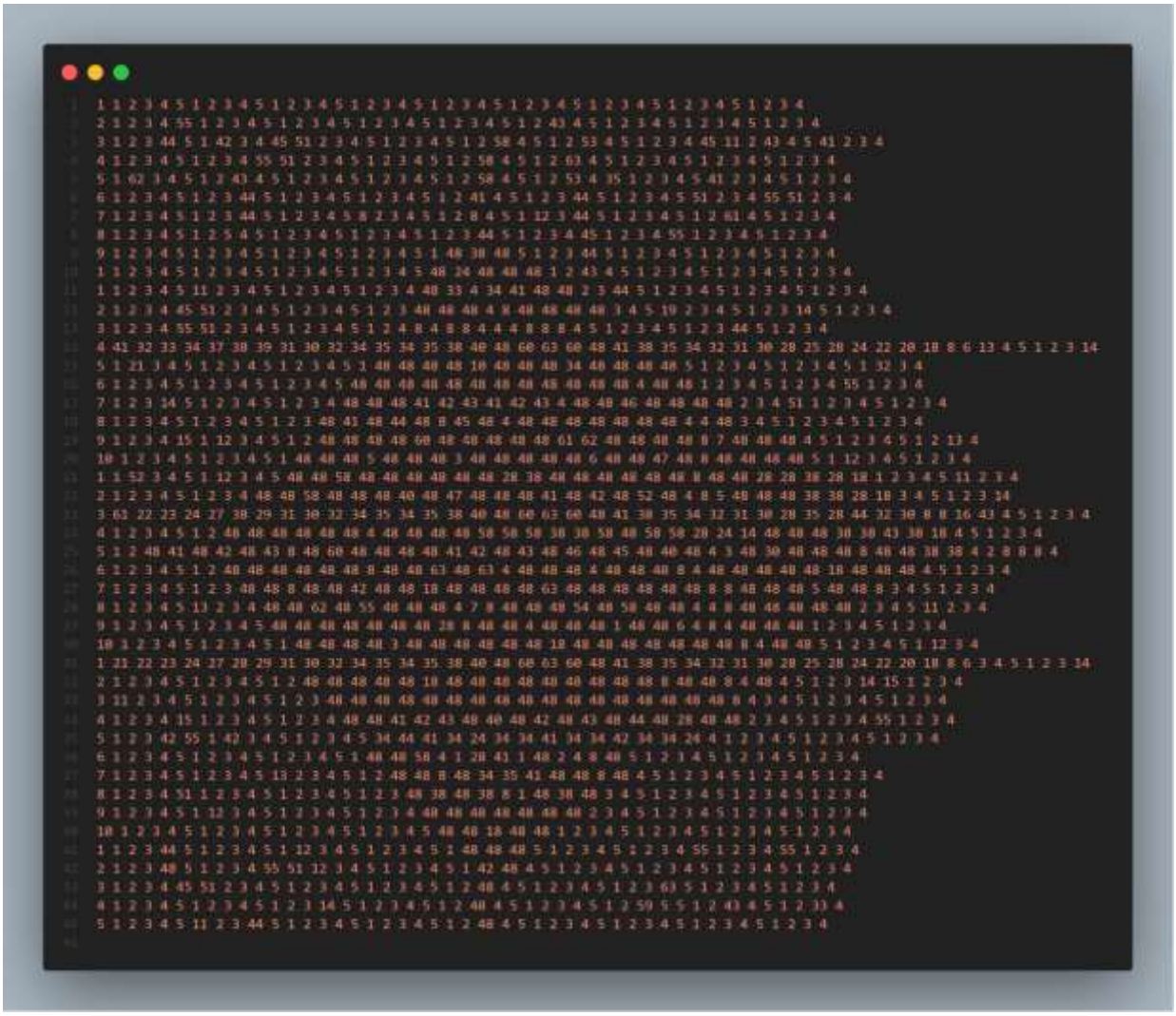

Figure 1 rfImg

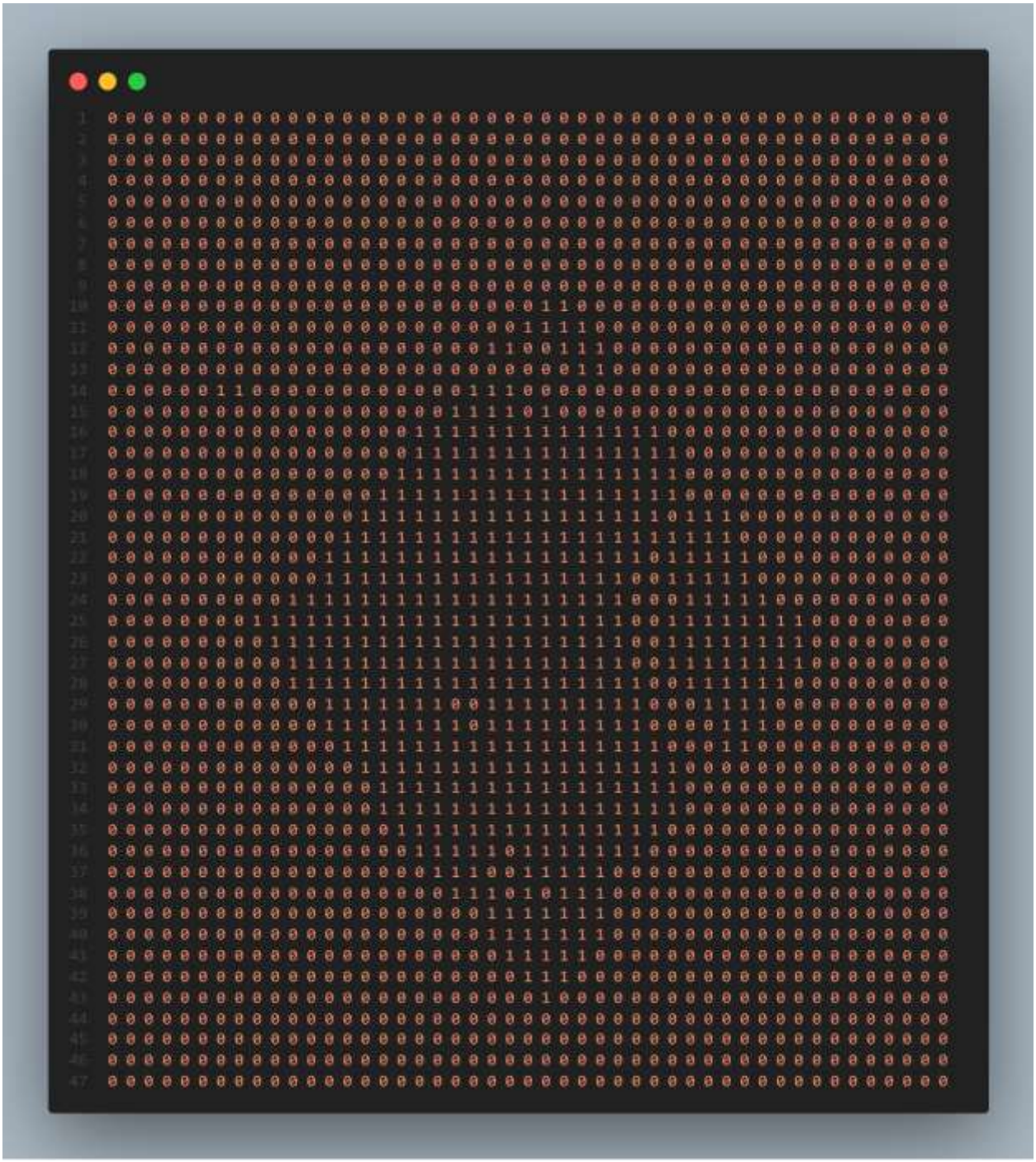

Figure 2 AvgOutImg(argv[4])

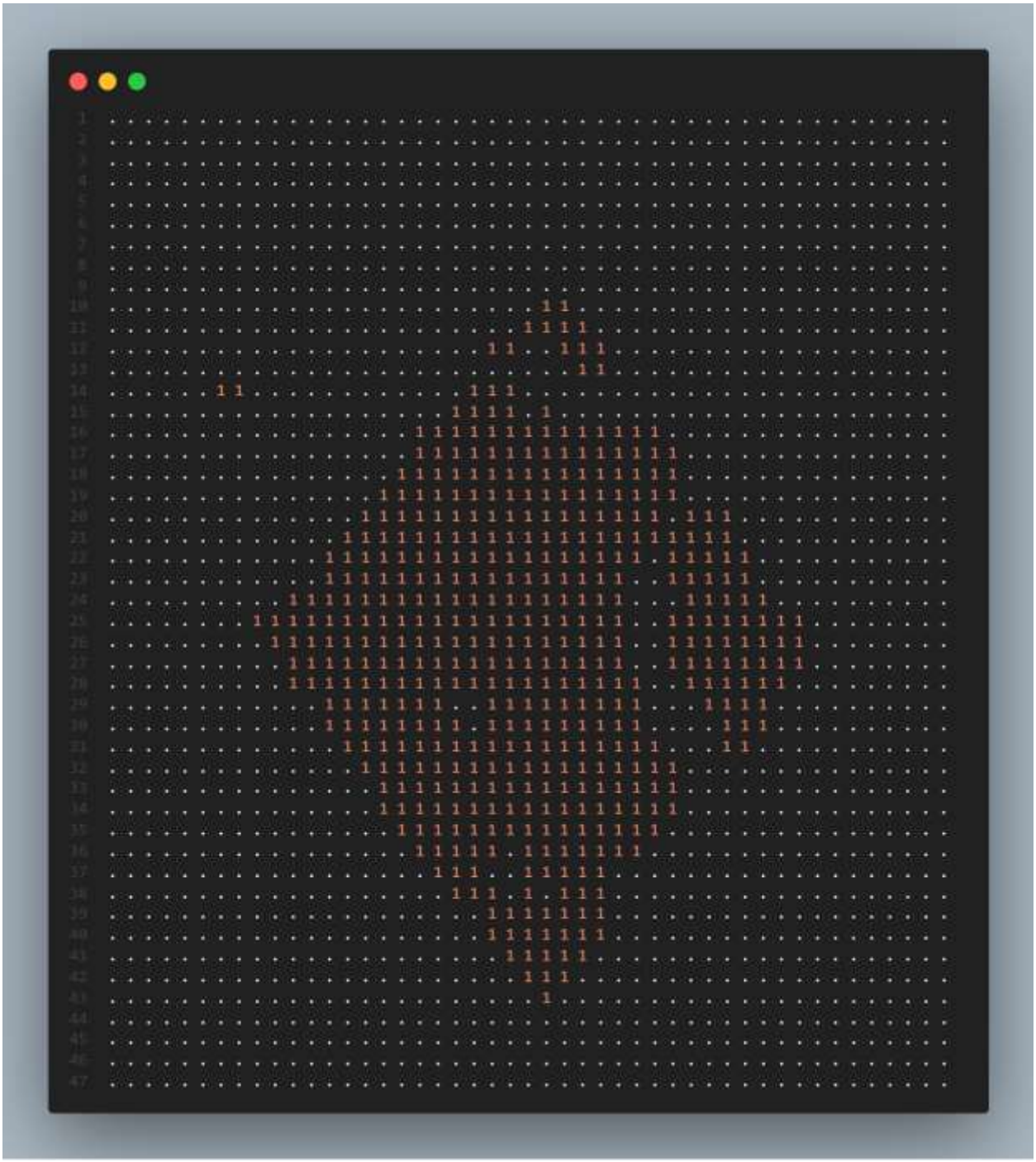

Figure 3 AvgPrettyPrint file

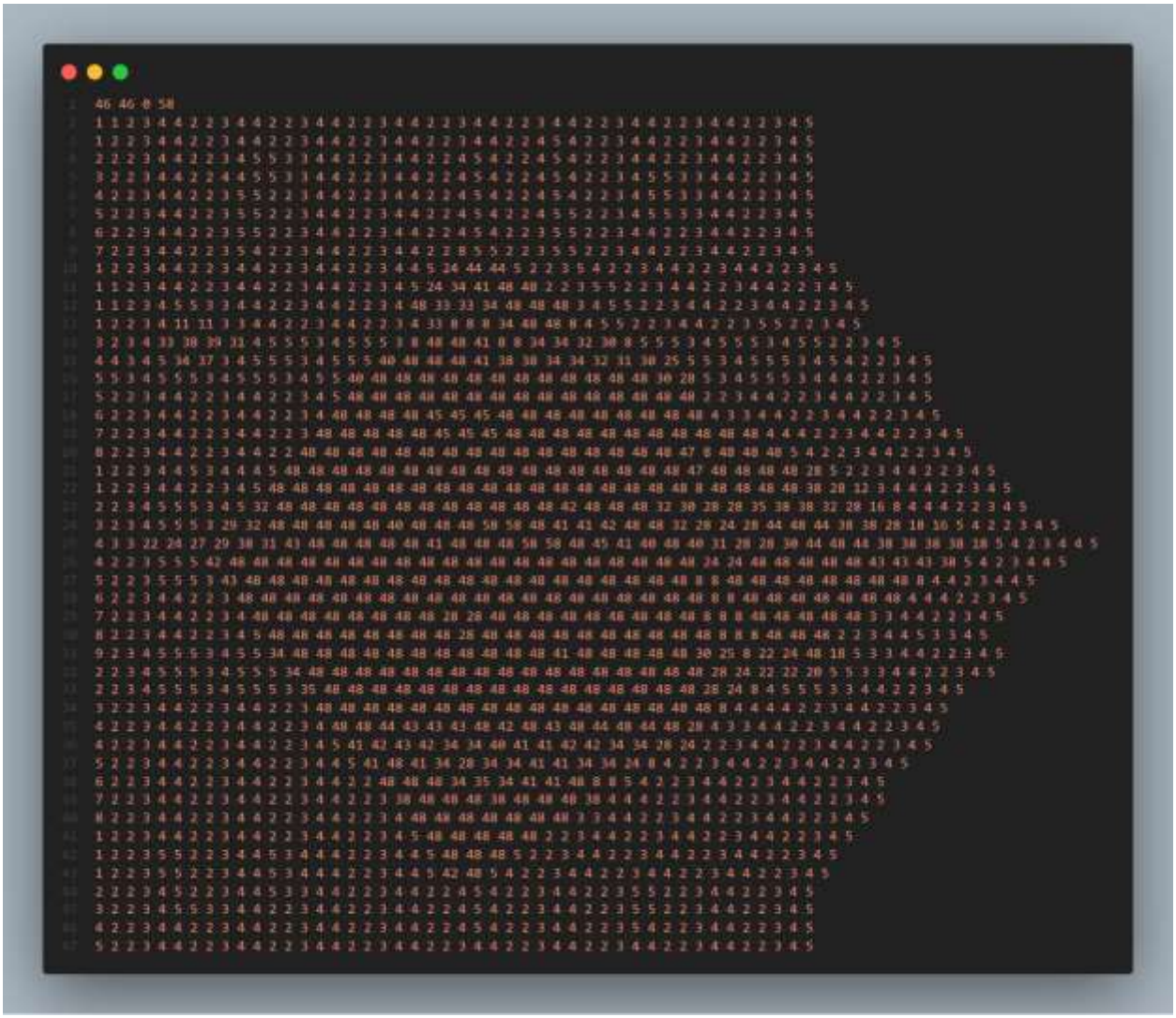

Figure 4 MedianOutImg

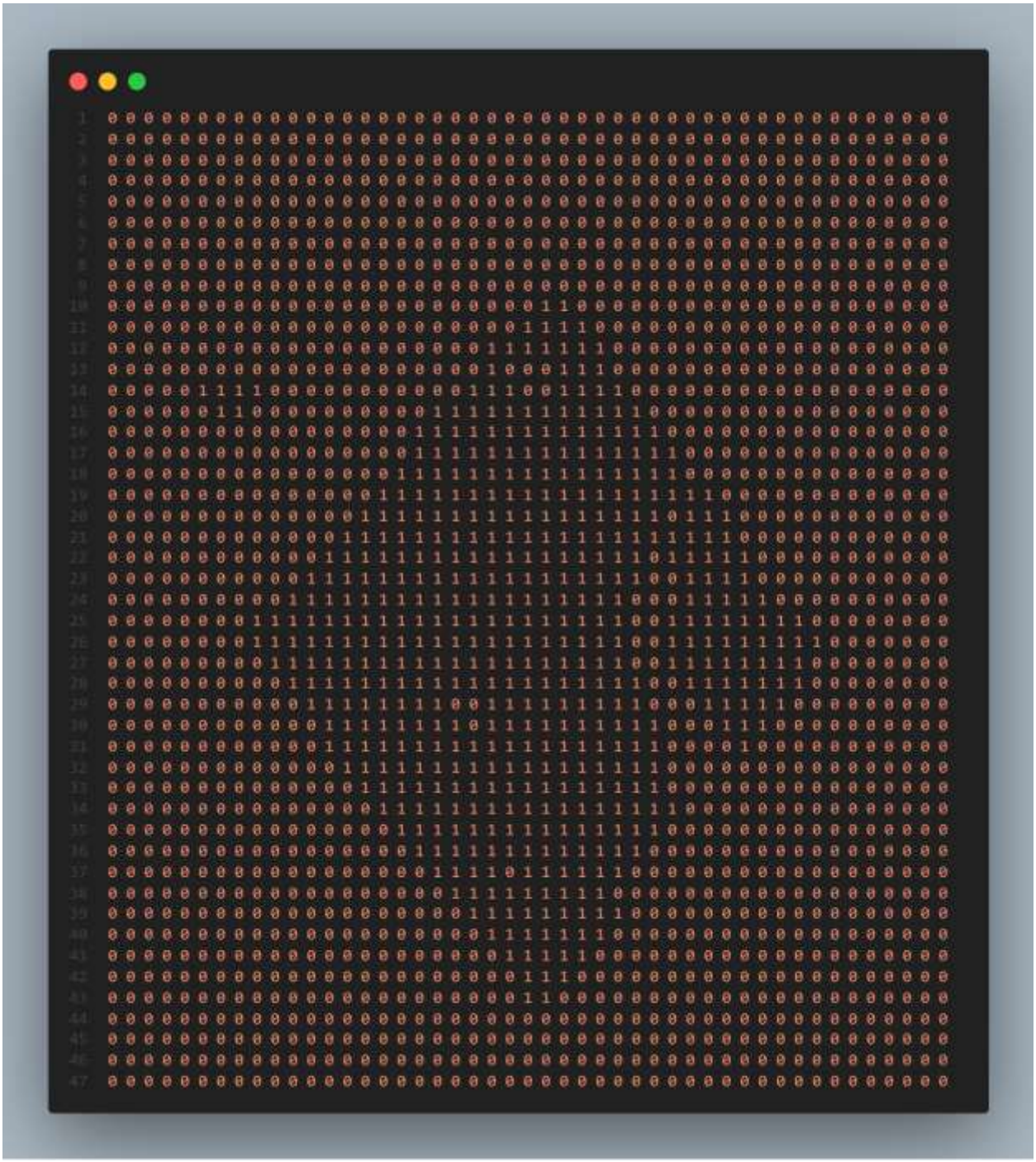

Figure 5 MedianThrImg

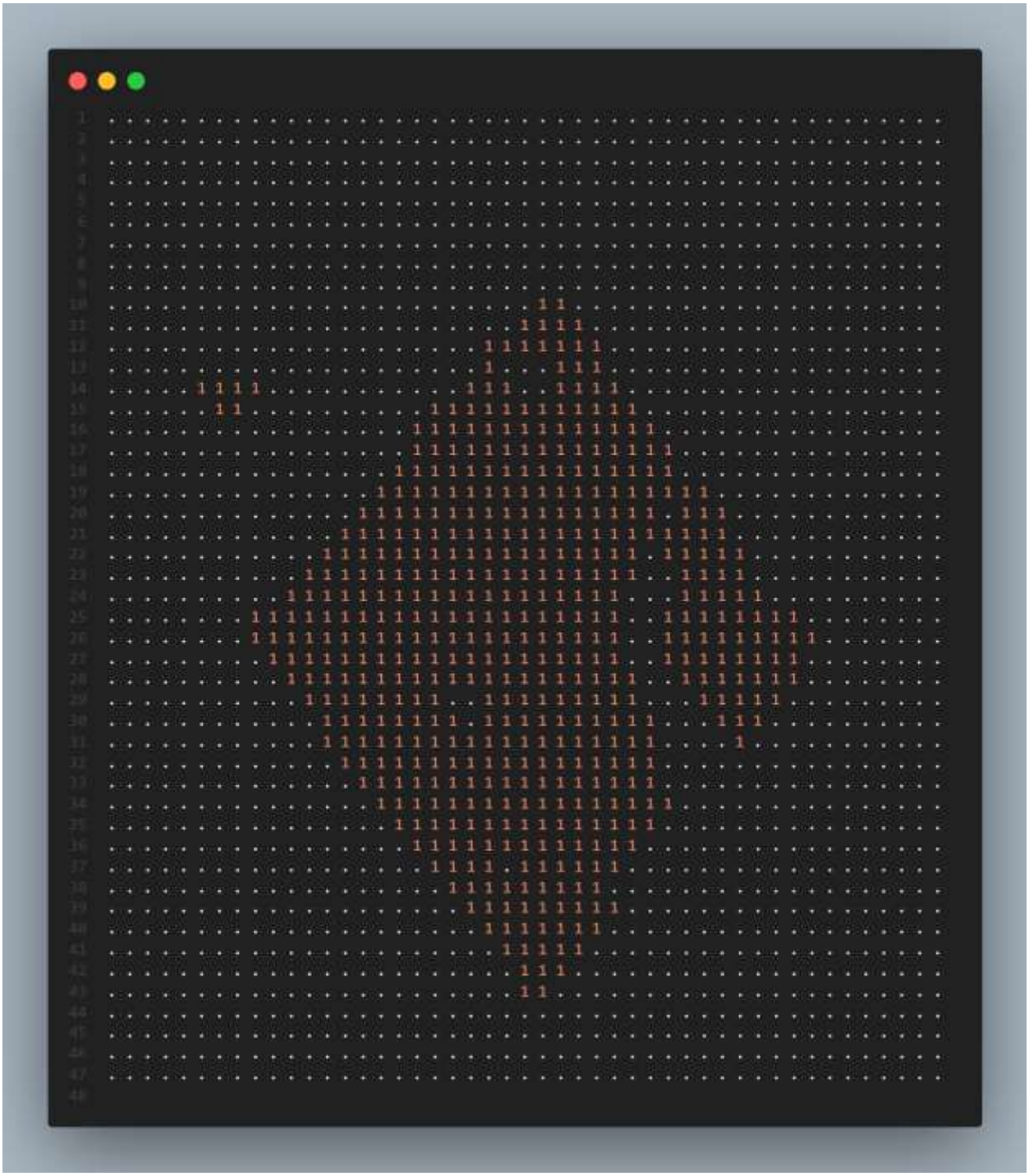

Figure 6 MedianPrettyPrint

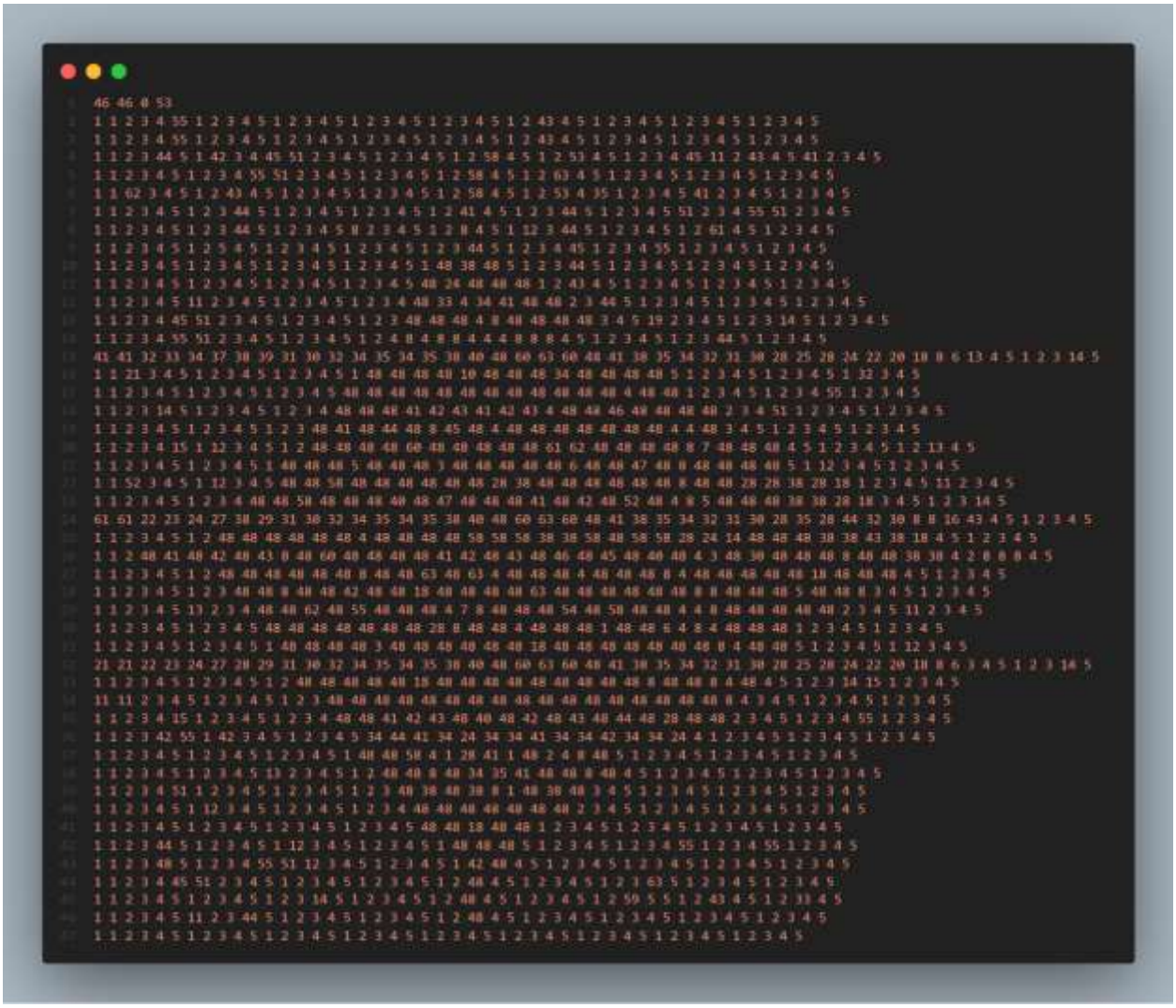

Figure 7 CPOutImg file

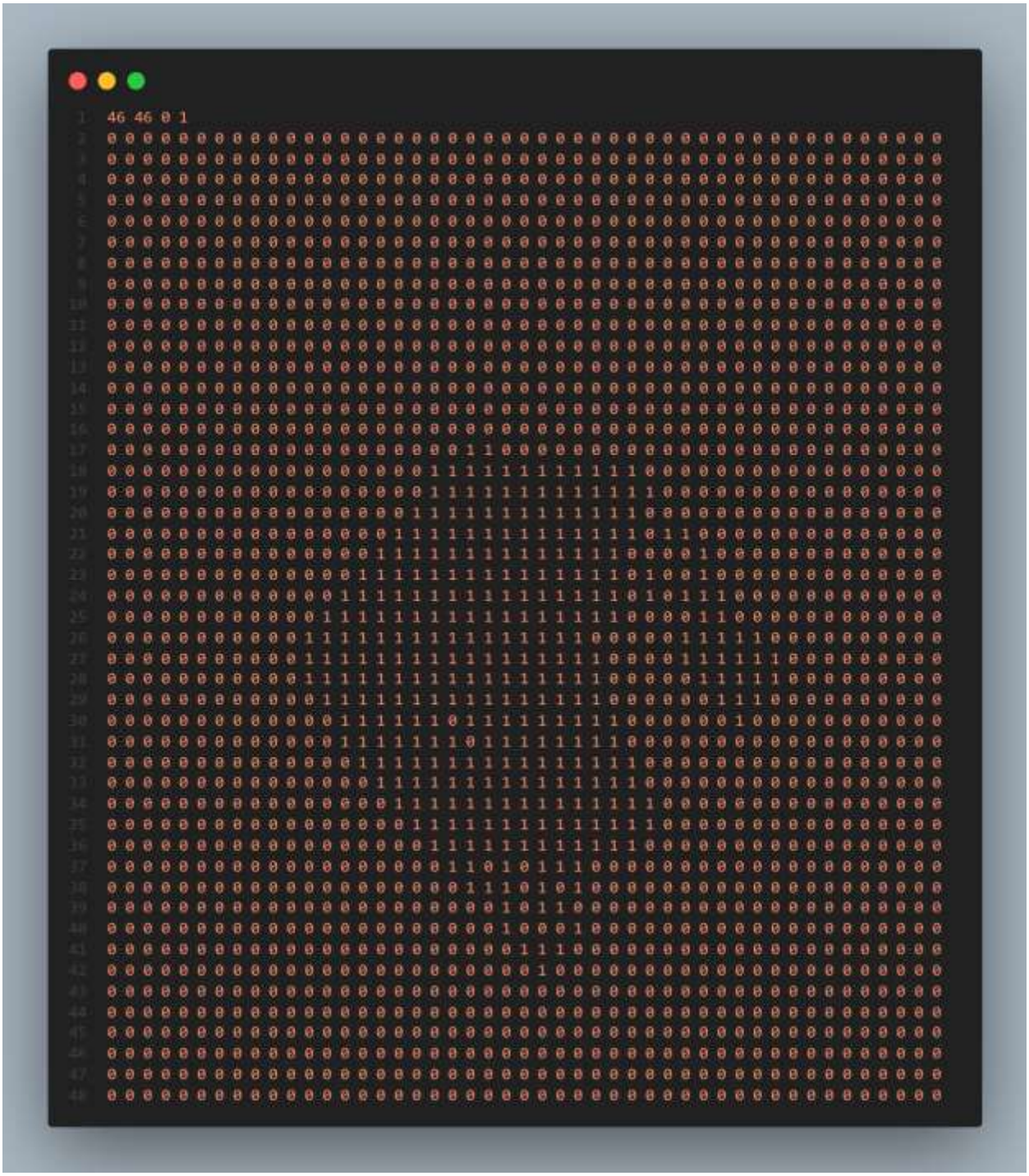

Figure 8 CPThrImg

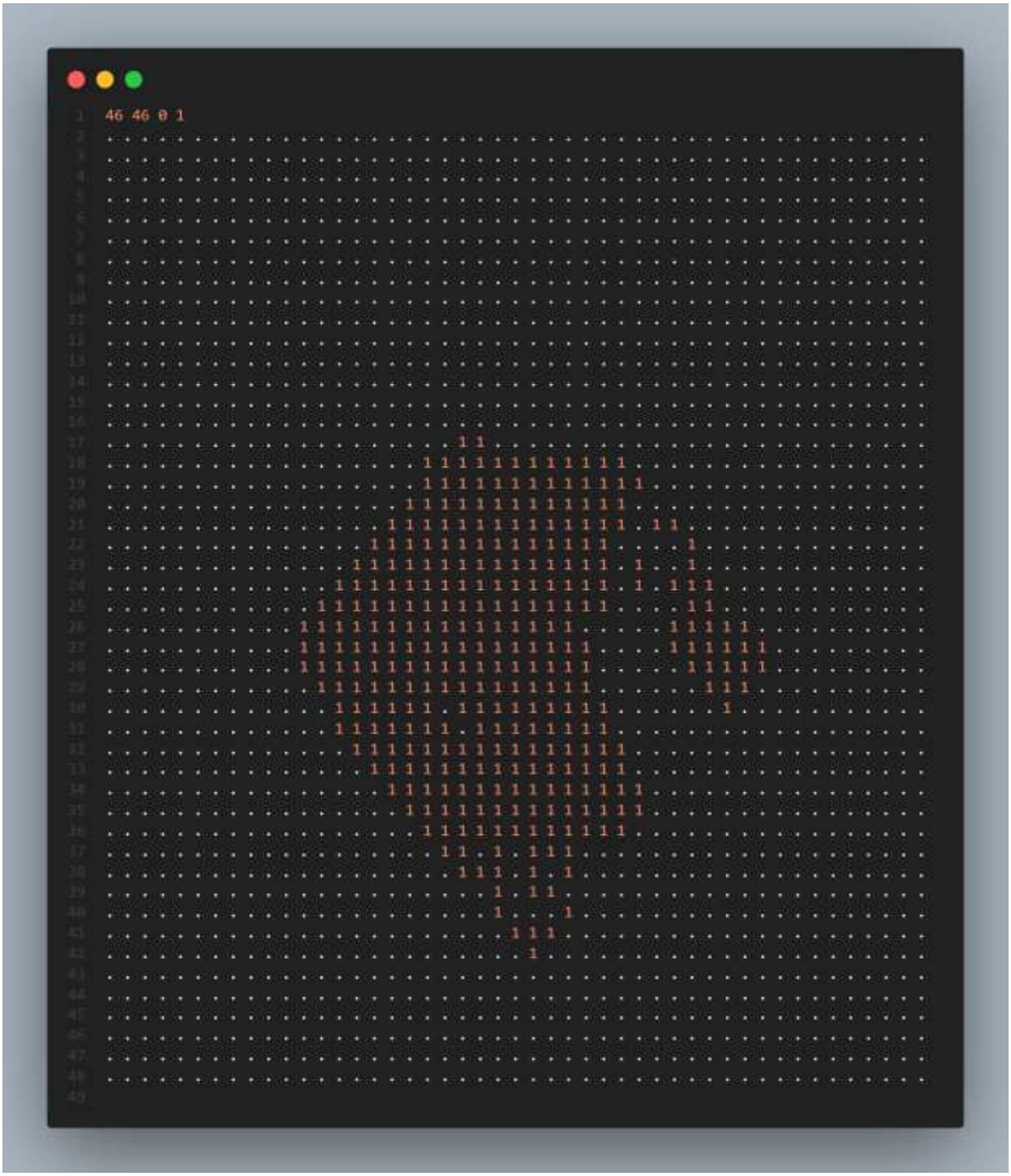

Figure 9 CPrettyPrint