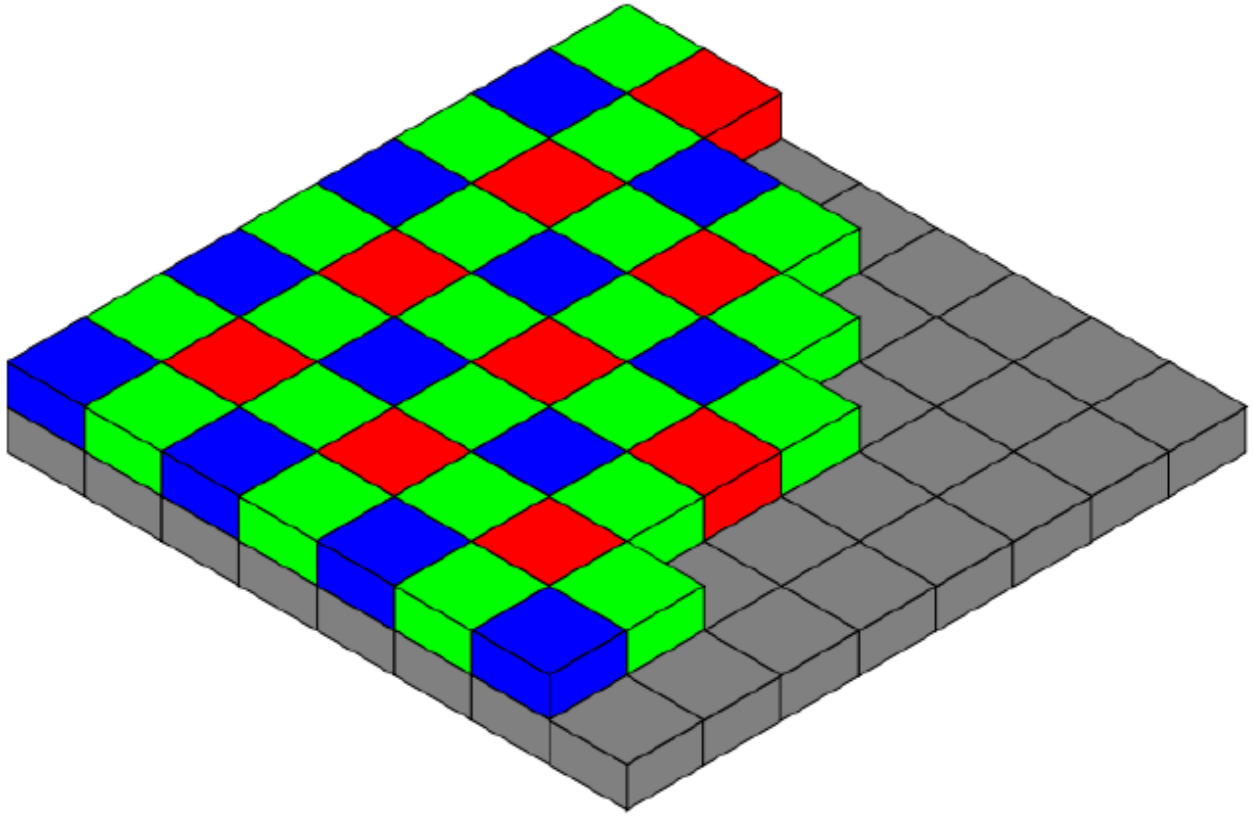
Bayer Interpolation using OpenMP



by Saidrasulkhon

ID: 22182232

Content

- Introduction
- Required task
- Proposed algorithm
- Result

Introduction

- What is Bayer filter?
  - It is a color filter array for arranging Red, Green and Blue color filters.
  - Invented by Bryce Bayer in 1976

The Bayer filter is almost universal on consumer digital cameras

Tasks performed

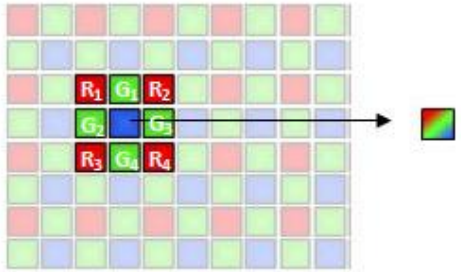
- Load the .raw format image
- Read the file and store in a one-dimensional array
- Allocate the memory space for image
- Convert 8-bit image values to 10-bit
- Separate each R,G and B channels
- Interpolate each channel
- Interpolate using OpenMP
- Convert to OpenCV Mat format
- Save the image
- Create the GUI using Qt

8 bit to 10 bit

```
void seq_data_copy(unsigned char *p, unsigned short *data, int size)
{
    int i = 0, j = 0;
    for (i, j; i < size; i += 4, j += 5)
    {
        data[i] = (p[j] << 2) + ((p[j + 4] >> 0) & 3);
        data[i + 1] = (p[j + 1] << 2) + ((p[j + 4] >> 2) & 3);
        data[i + 2] = (p[j + 2] << 2) + ((p[j + 4] >> 4) & 3);
        data[i + 3] = (p[j + 3] << 2) + ((p[j + 4] >> 6) & 3);
    }
}
```

The code was given that this code convert the 8-bit data to 10-bit

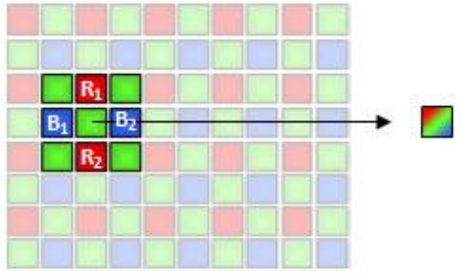
Image interpolation



Blue pixel:
Interpolation of green
& red color information

Red= $(R_1 + R_2 + R_3 + R_4) / 4$
Blue= Blue
Green= $(G_1 + G_2 + G_3 + G_4) /4$

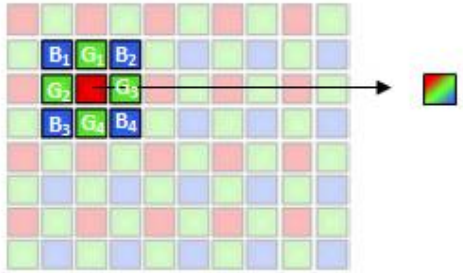
After separation image to individual channel I had 2D array of values of each channel



Green pixel:
Interpolation of blue
& red color information

Red= $(R_1 + R_2) / 2$
Blue= $(B_1 + B2_2) /2$
Green= Green

To calculate missing values of channels we must use formula of averaging



Red pixel:
Interpolation of blue
& green color information

Red= Red
Blue= $(B_1 + B_2 + B_3 + B_4) / 4$
Green= $(G_1 + G_2 + G_3 + G_4) /4$

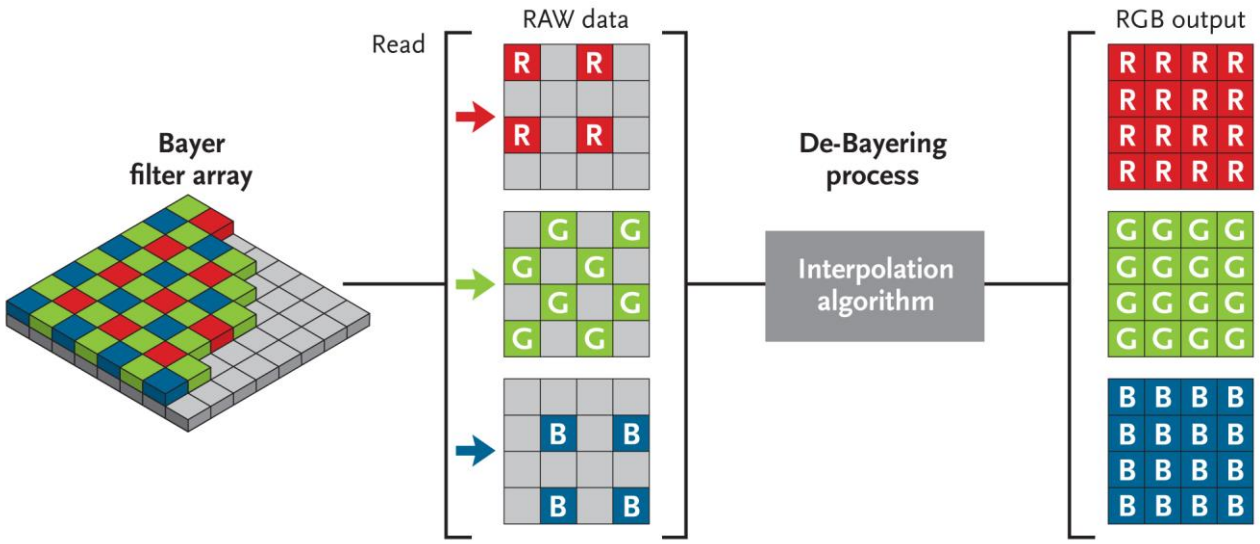
Proposed algorithm

```
for (int r = 0; r < height; r++)
{
    for (int c = 0; c < width; c++)
    {
        counter++;
        if ((c % 2 == 0) && (r % 2 == 0))
        {
            red[r][c] = data[counter];
        }
        if ((c % 2) == 1 && (r % 2) == 0)
        {
            green[r][c] = data[counter];
        }
        if ((c % 2) == 1 && (r % 2) == 1)
        {
            blue[r][c] = data[counter];
        }
        if ((c % 2) == 0 && (r % 2) == 1)
        {
            green[r][c] = data[counter];
        }
    }
}
```

Since I know the position of each R, G and B channels I separated them into three different unsigned short variables

This code is to separate channels from 10-bit unsigned short variable

Interpolation method



I must have filled missing pixel values by averaging the values of each channel

```
void interpolationRed(unsigned short ** arr, int width, int height) { ... }
void interpolationBlue(unsigned short ** arr, int width, int height) { ... }
void interpolationGreen(unsigned short ** arr, int width, int height) { ... }
```

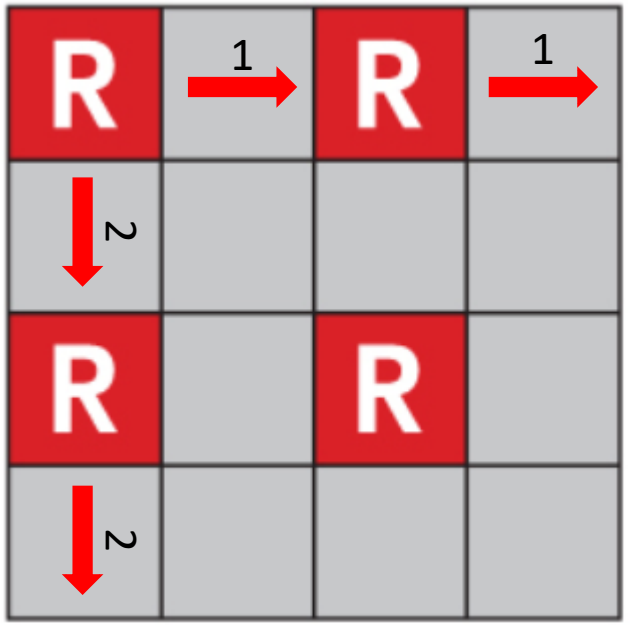
I used separate functions to interpolate each channel

Interpolation algorithm Red

```
void interpolationRed(unsigned short ** arr, int width, int height)
{//calculate column
    for (int r = 0; r < height; r += 2)
        for (int c = 0; c < width; c += 2) {
            if (c < width - 2)
            {
                arr[r][c + 1] = (arr[r][c] + arr[r][c + 2]) / 2;
            }
        }
    //calculate row
    for (int c = 0; c < width; c++)
        for (int r = 0; r < height; r+=2) {
            if (r < height - 2)
            {
                arr[r+1][c] = (arr[r][c] + arr[r+2][c]) / 2;
            }
            arr[height][c] = (arr[height-1][c]);

        }
}
```

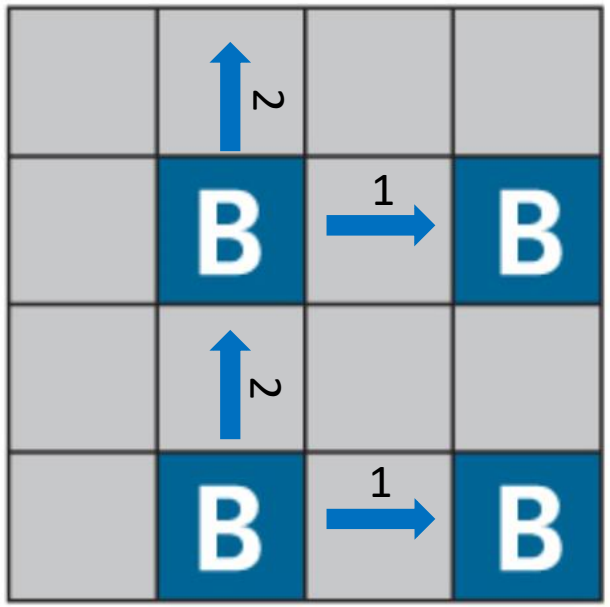
First, I calculate the values by column and then I calculated values by row.



Interpolation algorithm Blue

```
void interpolationBlue(unsigned short ** arr, int width, int height)
{
    //calculate column
    for (int r = 1; r < height; r++)
    {
        for (int c = 1; c < width; c+=2)
        {
            if (c < width - 2)
            {
                arr[r][c + 1] = (arr[r][c] + arr[r][c + 2]) / 2;
            }
            arr[r - 1][0] = arr[r][1];
        }
    }
    //calculate row
    for (int c = 1; c < width; c++)
    {
        for (int r = 1; r < height; r+=2)
        {
            if (r < height - 2)
            {
                arr[r + 1][c] = (arr[r][c] + arr[r + 2][c]) / 2;
            }
                arr[0][c] = (arr[1][c]);
        }
    }
}
```

I used almost same calculation algorithm for blue with some minor changes



Same as red I calculated first column then row

Interpolation algorithm Green

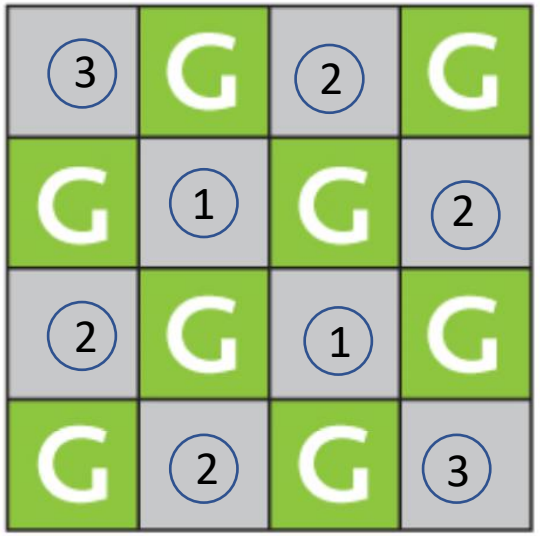
```
void interpolationGreen(unsigned short ** arr, int width, int height)
{

    for (int r = 1; r < height; r+=2)
    {
        for (int c = 1; c < width; c+=2)
        {
        if (c<width-1 && c > 0 && r > 0 && r< height-1)
            {
                arr[r][c] = (arr[r - 1][c] + arr[r][c - 1] + arr[r + 1][c] + arr[r][c + 1]) / 4;
            }
        }
    }

    for (int c = 0; c < width; c += 2)
    {
        for (int r = 0; r < height; r += 2)
        {
            if (c < width - 1 && c > 0 && r > 0 && r < height - 1)
            {
                arr[r][c] = (arr[r - 1][c] + arr[r][c - 1] + arr[r + 1][c] + arr[r][c + 1]) / 4;
            }

        }
    }
    for (int c = 0; c < width; c+=2)
    {
        for (int r = 0; r < height; r += 2)
        {
            if (c > 0 && r > 0 && c < width - 1 && r < height - 1)
            {
                arr[0][r] = (arr[0][r - 1] + arr[0][r + 1] + arr[1][r]) / 3;
            }
        }
    }
    for (int r = 0; r < height; r += 2)
    {
        for (int c = 0; c < width; c += 2)
        {
            if (c > 0 && r > 0 && c < width - 1 && r < height - 1)
            {
                arr[r][0] = (arr[r-1][0] + arr[r+1][0] + arr[r][1]) / 3;
            }
            if (c == 0 && r == 0)
            {
                arr[r][c] = (arr[r + 1][c] + arr[r][c + 1]) / 2;
            }
        }
    }
```

To interpolate green channel I had to calculate it using different formula



First, I calculated the cases where I should calculate average of four border values then I calculated corners with border line

OpenCV format conversion

```
Vec3w vec;
Mat matrix = Mat(Size(width, height), CV_16UC3);
for (int r = 0; r < height; r++) {
    for (int c = 0; c < width; c++) {
        vec[0] = blue[r][c] * 64;
        vec[1] = green[r][c] * 64;
        vec[2] = red[r][c] * 64;
        matrix.at<Vec3w>(r, c) = vec;
    }
}
```

This code is enabling me to convert the unsigned short array to Mat format to see the expected image.

Processing time Serial VS Parallel

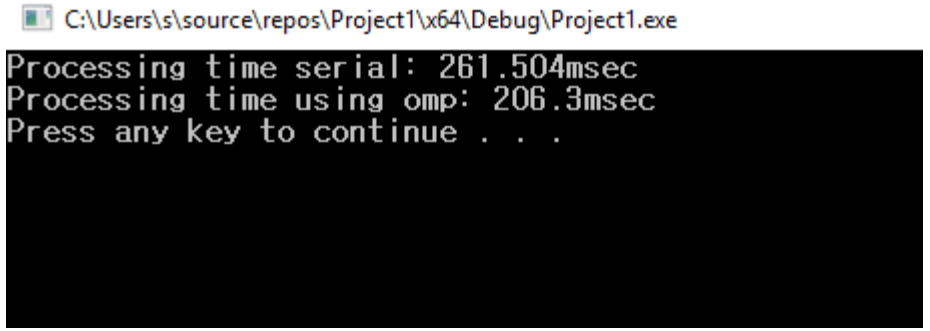
Intel Core i5-7600 @ 3.50GHz

Description: *Intel HD Graphics 630*
Socket: *FCLGA1151*
Clockspeed: *3.5 GHz*
Turbo Speed: *4.1 GHz*
No of Cores: *4*
Typical TDP: *65 W*

```
void interpolationBlue(unsigned short ** arr, int width, int height)
{//calculate column
#pragma omp parallel sections
    {
        #pragma omp section
        {
            for (int r = 1; r < height; r++)
            {
                for (int c = 1; c < width; c += 2)
                {
                    if (c < width - 2)
                    {
                        arr[r][c + 1] = (arr[r][c] + arr[r][c + 2]) / 2;
                    }
                    arr[r - 1][0] = arr[r][1];
                }
            }
        }

        //calculate row
        #pragma omp section
        {
            for (int c = 1; c < width; c++)
            {
                for (int r = 1; r < height; r += 2)
                {
                    if (r < height - 2)
                    {
                        arr[r + 1][c] = (arr[r][c] + arr[r + 2][c]) / 2;
                    }
                    arr[0][c] = (arr[1][c]);
                }
            }
        }

    }
}
```



Processing time wasn't much different when I used omp and serial.
For parallel processing I used omp sections to calculate each for loop in different sections
omp_set_num_threads(4).

Result image