ALGORITMI ZA OBRADU SLIKA

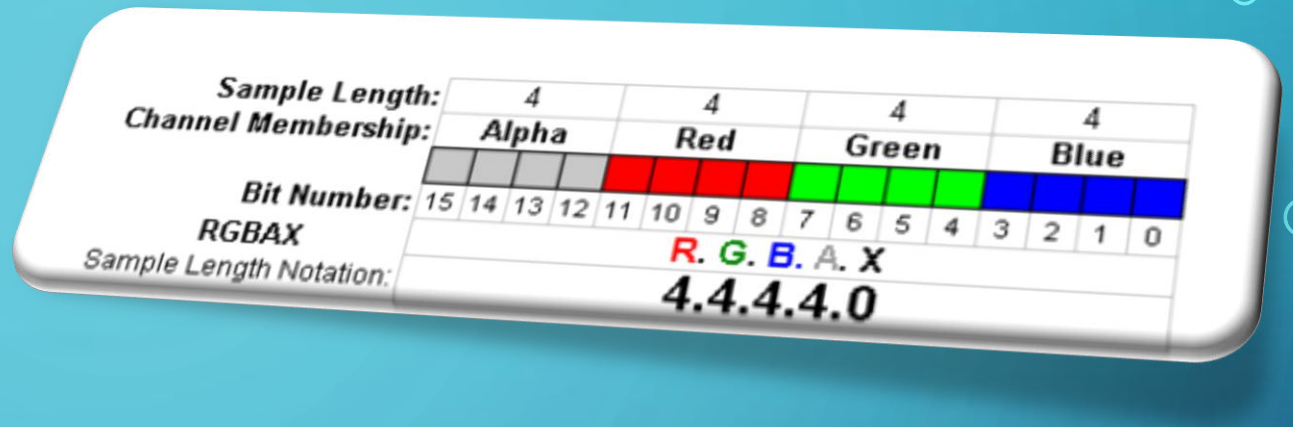
POSVETLJIVANJE

GAMA KOREKCIJA

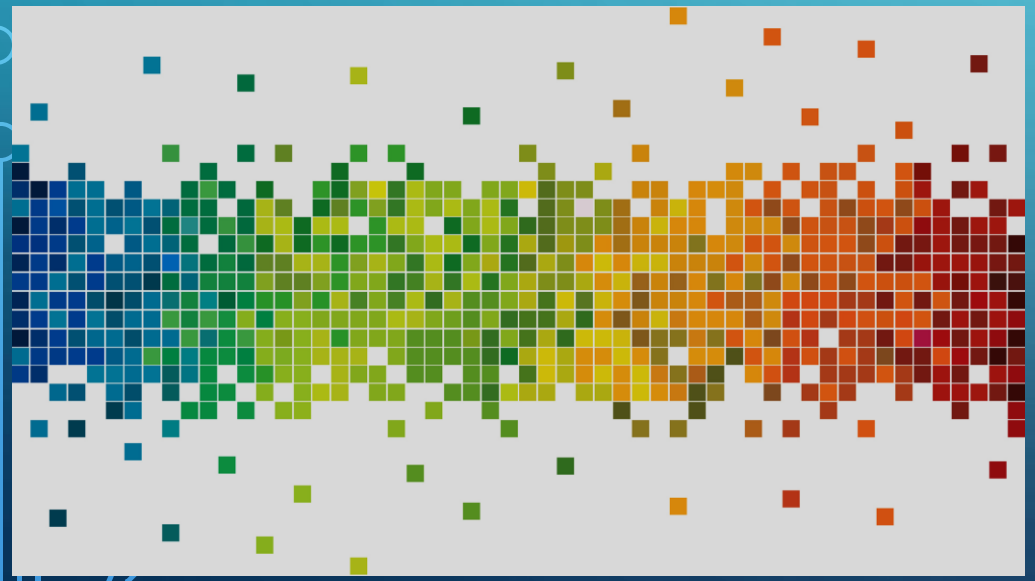
IRENA SAVIĆ 318/2013

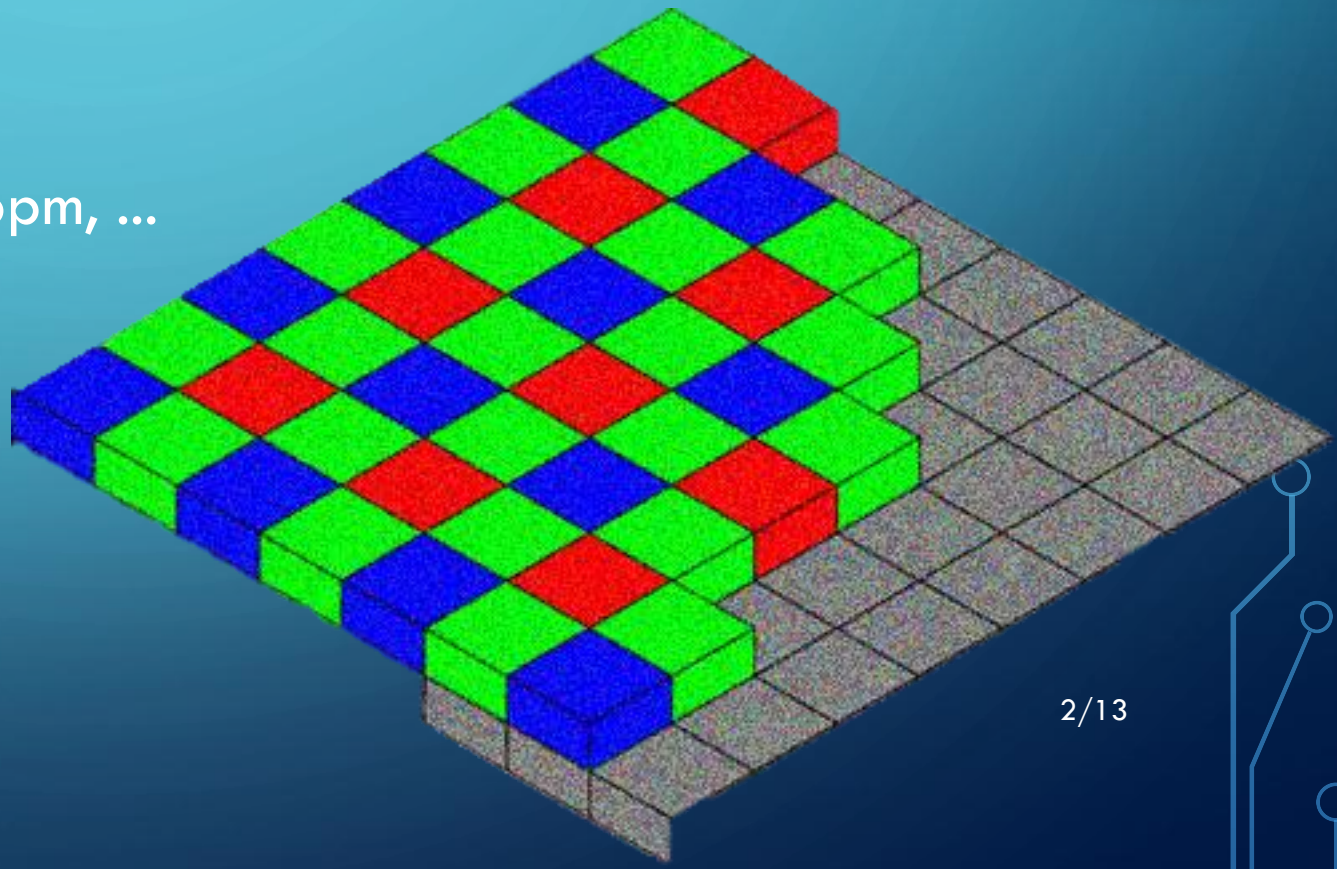
VELJKO VRANIĆ 54/2013

OBRADA SLIKA

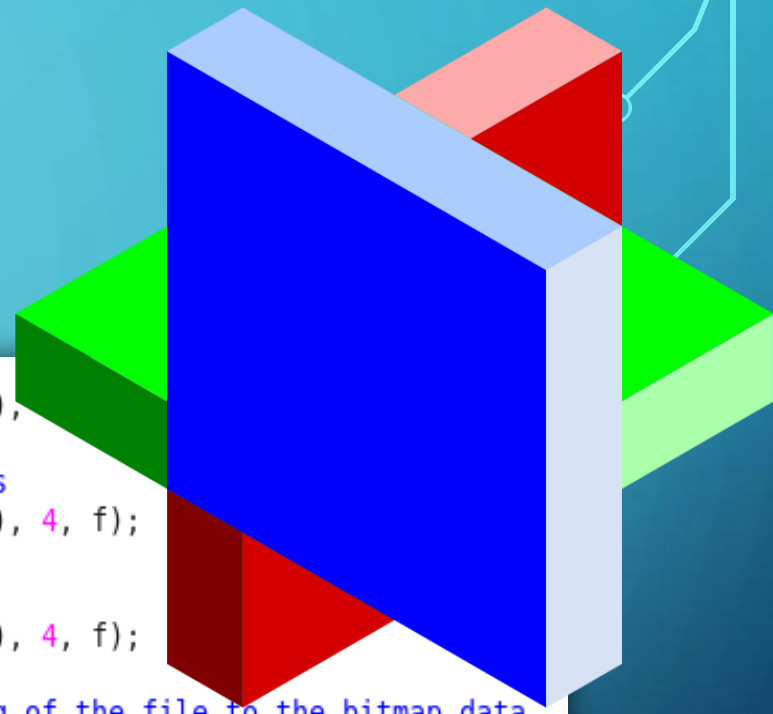


- Vrlo se lako paralelizuje
- RGB format slika
- Različiti formati: jpg, png, ppm, bpm, ...





```
void write_BITMAPFILEHEADER(FILE* f, unsigned int width, unsigned int heigh
  {
        unsigned int offset = BITMAPFILEHEADER_SIZE + BITMAPINFOHEADER_SIZE
        unsigned char arrayToWrite[4];

        //must always be set to 'BM' to declare that this is a .bmp-file
        arrayToWrite[0] = 'B';
        arrayToWrite[1] = 'M';
    fwrite(arrayToWrite, sizeof(char), 2, f);

        //the size of the file in bytes
        arrayToWrite[0] = (unsigned char)(fileSize);
        arrayToWrite[1] = (unsigned char)(fileSize >> 8);
        arrayToWrite[2] = (unsigned char)(fileSize >> 16);
        arrayToWrite[3] = (unsigned char)(fileSize >> 24);
        fwrite(arrayToWrite, sizeof(char), 4, f);

        //reserved, must always be set to zero
        arrayToWrite[0] = 0;
        arrayToWrite[1] = 0;
        arrayToWrite[2] = 0;
        arrayToWrite[3] = 0;
        fwrite(arrayToWrite, sizeof(char), 4, f);

        //the offset from the beginning of the file to the bitmap
        arrayToWrite[0] = (unsigned char)(offset);
```



```
        //BM
        fread(arrayToRead, sizeof(char),

        //the size of the file in bytes
        fread(arrayToRead, sizeof(char), 4, f);

        //reserved
        fread(arrayToRead, sizeof(char), 4, f);

        //the offset from the beginning of the file to the bitmap data
        fread(arrayToRead, sizeof(char), 4, f);
        offset = (unsigned int)arrayToRead[0];
        offset = offset | ((unsigned int)arrayToRead[1] << 8);
        offset = offset | ((unsigned int)arrayToRead[2] << 16);
        offset = offset | ((unsigned int)arrayToRead[3] << 24);

        //the size of the BITMAPINFOHEADER structure, in bytes
    fread(arrayToRead, sizeof(char), 4, f);

        //the width of the image, in pixels
    fread(arrayToRead, sizeof(char), 4, f);
        *width = (unsigned int)arrayToRead[0];
        *width = *width | ((unsigned int)arrayToRead[1] << 8);
        *width = *width | ((unsigned int)arrayToRead[2] << 16);
        *width = *width | ((unsigned int)arrayToRead[3] << 24);
```

ALGORITAM POSVETLJIVANJA

- Osobina vizuelne percepcije u kojoj se čini da izvor zrači ili reflektuje svetlost
- Percepcija je izazvana osvetljenošću vizuelnog cilja

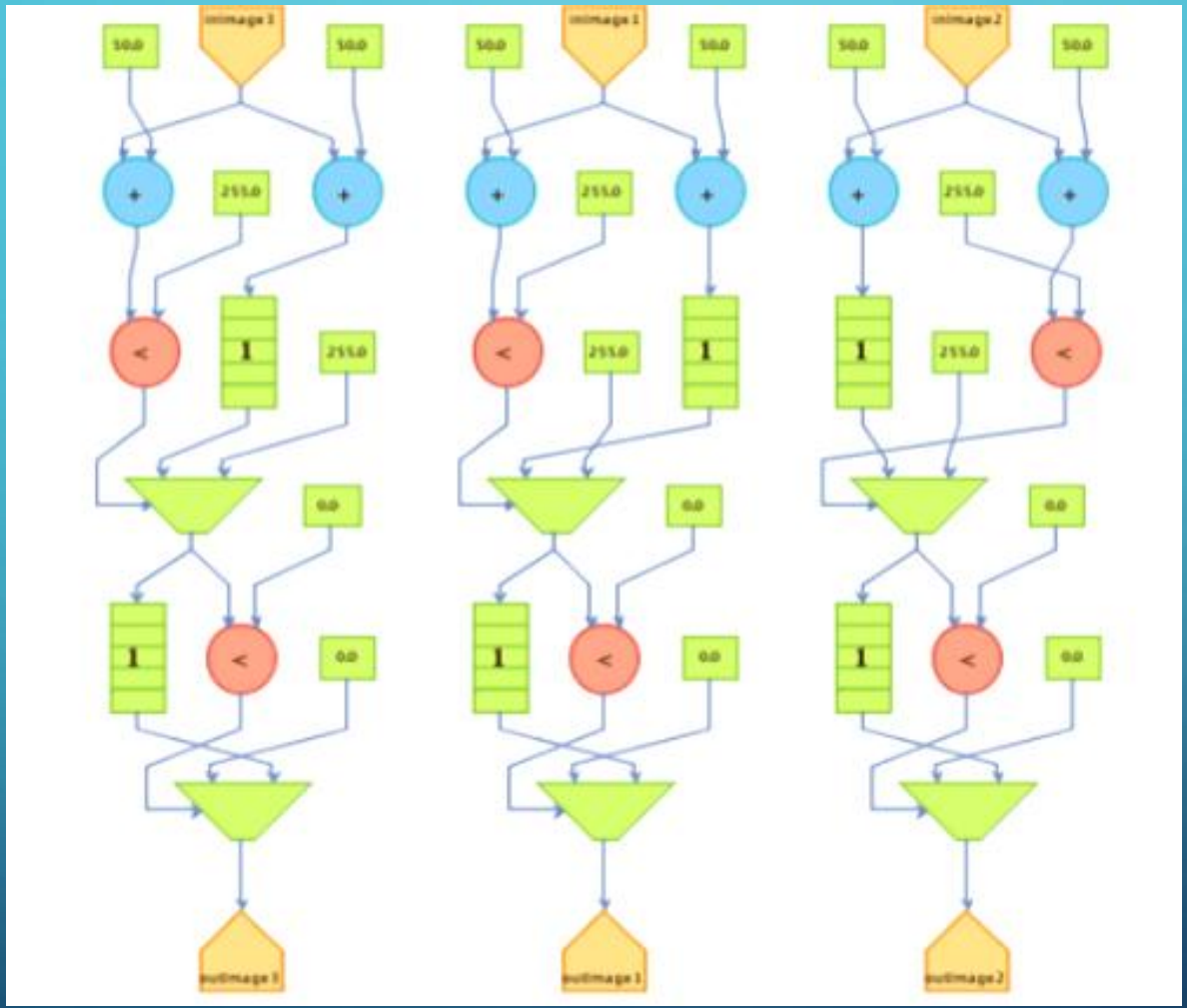
```
colour = GetPixelColour(x, y)
newRed   = Truncate(Red(colour)   + brightness)
newGreen = Truncate(Green(colour) + brightness)
newBlue  = Truncate(Blue(colour)  + brightness)
PutPixelColour(x, y) = RGB(newRed, newGreen, newBlue)

If value < 0 Then value = 0
  If value > 255 Then value = 255
```

```
int cnt = 0;

for (int i = 0; i < width*height*(BITS_PER_PIXEL/8); i+
  if (i%3 == 0)
    array1[cnt] = (int32_t)bitmapBits[i];
  else
    if (i%3 == 1)
      array2[cnt] = (int32_t)bitmapBits[i];
    else
      array3[cnt++] = (int32_t)bitmapBits[i];

int dataSize = width * height * sizeof(int);
// Allocate a buffer for the output image
int32_t *outImage1 = malloc(dataSize);
int32_t *outImage2 = malloc(dataSize);
int32_t *outImage3 = malloc(dataSize);

brightness_cpu(bitmapBits_cpu,width,height);

printf("Running Kernel.\n");

Brightness(width * height, array1, array2, array3, out

double max_time;
stopTime(max_time);
printf("Maxeler  - %.2f\n", max_time * 1e-3);
printf("Saving image.\n");
cnt = 0;

for (int i = 0; i < width*height*(BITS_PER_PIXEL/8); i++)
  if (i%3 == 0)
    bitmapBits[i] = (char)outImage1[cnt];
  else
    if (i%3 == 1)
      bitmapBits[i] = (char)outImage2[cnt];
    else
      bitmapBits[i] = (char)outImage3[cnt++];
```

```
class BrightnessKernel extends Kernel {

    private static final DFEType type = dfeInt(32);

    BrightnessKernel(KernelParameters parameters) {
        super(parameters);

        DFEVar input1 = io.input("inImage1", type);
        DFEVar input2 = io.input("inImage2", type);
        DFEVar input3 = io.input("inImage3", type);

        int bc = 50;

        DFEVar result1 = input1 + bc < 255 ? input1 + bc : 255;
               result1 = result1 < 0 ? 0 : result1;
        DFEVar result2 = input2 + bc < 255 ? input2 + bc : 255;
                result2 = result2 < 0 ? 0 : result2;
        DFEVar result3 = input3 + bc < 255 ? input3 + bc : 255;
                result3 = result3 < 0 ? 0 : result3;

        io.output("outImage1", result1, type);
        io.output("outImage2", result2, type);
        io.output("outImage3", result3, type);
    }
}
```

```
package brightness;

import com.maxeler.maxcompiler.v2.build.EngineParameters;

class BrightnessManager {

    public static void main(String[] args) {
        Manager manager = new Manager(new EngineParameters(args));
        Kernel kernel = new BrightnessKernel(manager.makeKernelParameters());
        manager.setKernel(kernel);
        manager.setIO(IOType.ALL_CPU);
        manager.createSLiCinterface();
        manager.build();
    }
}
```

POSVETLJIVANJE - GRAF



GAMA KOREKCIJA

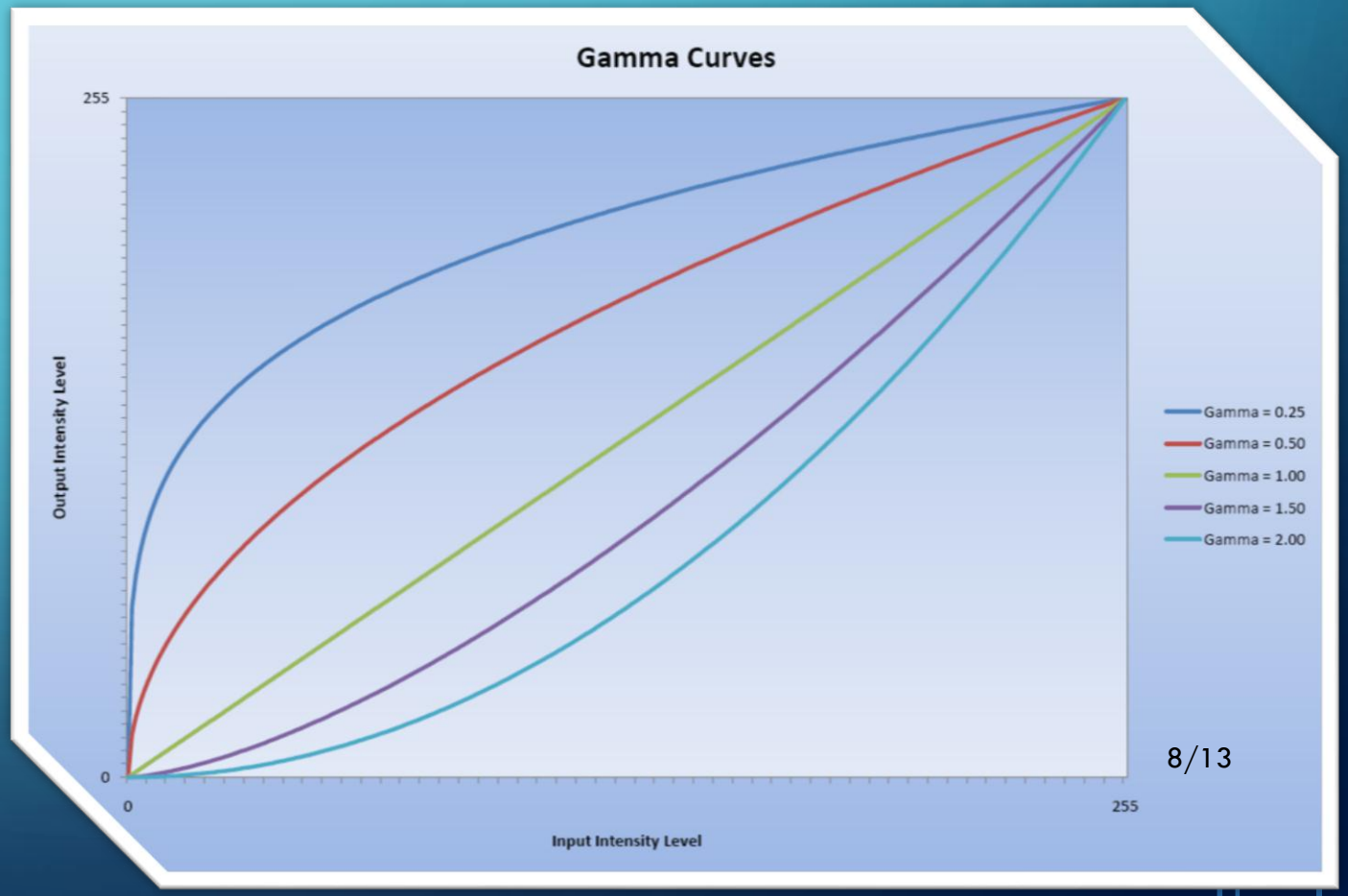
Gama – relacija ulazne i izlazne vrednosti piksela

Utiče na vrednost RGB komponenti

$$I' = 255 \times \left(\frac{I}{255}\right)^{\gamma}$$

Recipročna vrednost – gama korekcija

$$I' = 255 \times \left(\frac{I}{255}\right)^{1/\gamma}$$



ALGORITAM - GAMA KOREKCIJA

gammaCorrection = 1 / gamma

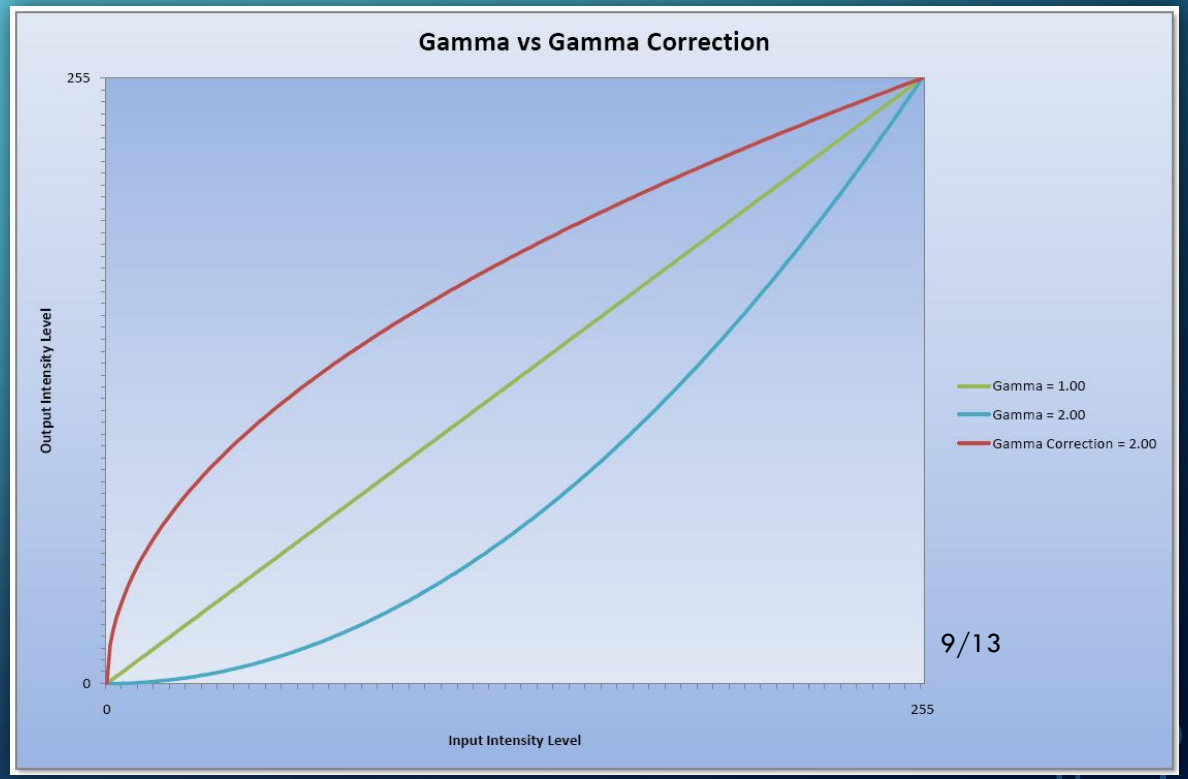
colour = GetPixelColour(x, y)

newRed = 255 * (Red(colour) / 255) ^ gammaCorrection

newGreen = 255 * (Green(colour) / 255) ^ gammaCorrection

newBlue = 255 * (Blue(colour) / 255) ^ gammaCorrection

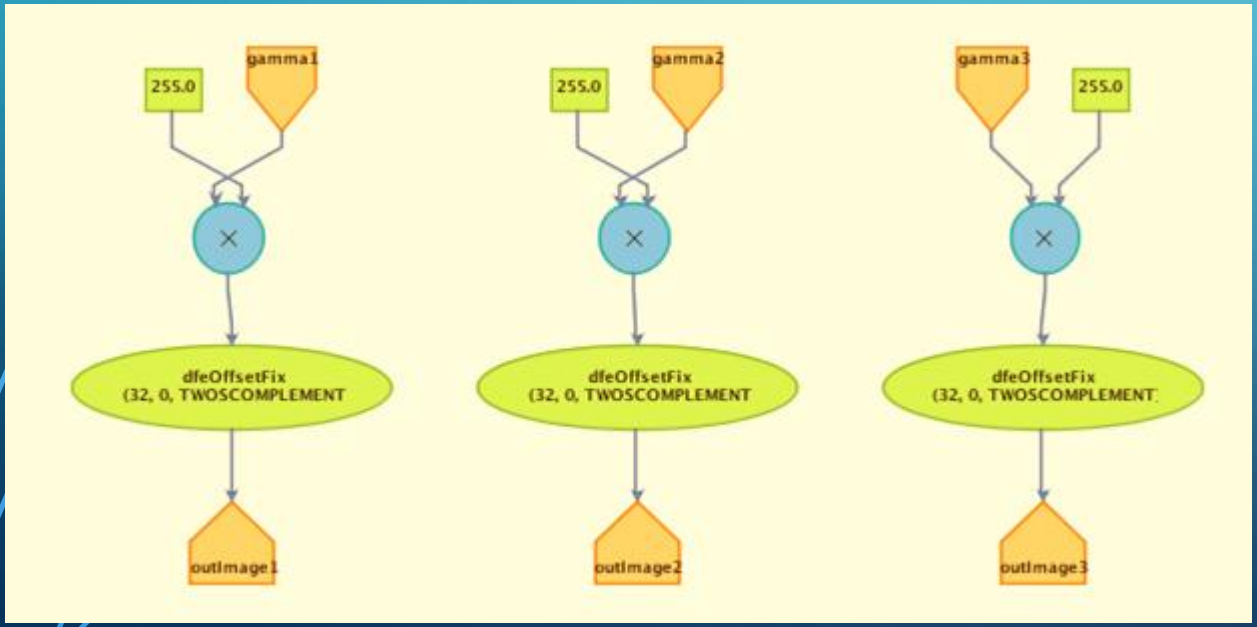
PutPixelColour(x, y) = RGB(newRed, newGreen, newBlue)



```
class BrightnessKernel extends Kernel {

        private static final DFEType typeInt = dfeInt(32);
        private static final DFEType typeFloat = dfeFloat(8,24);

        BrightnessKernel(KernelParameters parameters) {
                super(parameters);



DFEVar input1 = io.input("gamma1", typeFloat);
DFEVar input2 = io.input("gamma2", typeFloat);
DFEVar input3 = io.input("gamma3", typeFloat);


DFEVar result1 = input1 * constant.var(255.0);
DFEVar result2 = input2 * constant.var(255.0);
DFEVar result3 = input3 * constant.var(255.0);

io.output("outImage1", result1.cast(typeInt), typeInt);
io.output("outImage2", result2.cast(typeInt), typeInt);
io.output("outImage3", result3.cast(typeInt), typeInt);
```

```
    // Allocate a buffer for the output image
    int32_t *outImage1 = malloc(dataSize);
    int32_t *outImage2 = malloc(dataSize);
    int32_t *outImage3 = malloc(dataSize);


    printf("Running Kernel.\n");
    Brightness(width * height, gamma1, gamma2, gamma3, outImage1,outImage2, outImage3

    printf("Saving image.\n");

    cnt = 0;

    for (int i = 0; i < width*height*(BITS_PER_PIXEL/8); i++)
  if (i%3 == 0)
        bitmapBits[i] = (char)outImage1[cnt];
  else
        if (i%3 == 1)
          bitmapBits[i] = (char)outImage2[cnt];
            else
              bitmapBits[i] = (char)outImage3[cnt++];

    FILE* out = fopen("lennyResult.bmp", "wb");

write_BITMAPFILEHEADER(out, width, height);
```

```
package brightness;

import com.maxeler.maxcompiler.v2.build.EngineParameters;
import com.maxeler.maxcompiler.v2.kernelcompiler.Kernel;
import com.maxeler.maxcompiler.v2.managers.standard.Manager;
import com.maxeler.maxcompiler.v2.managers.standard.Manager.IOType;

class BrightnessManager {

        public static void main(String[] args) {
                Manager manager = new Manager(new EngineParameters(args));
                Kernel kernel = new BrightnessKernel(manager.makeKernelParameters());
                manager.setKernel(kernel);
                manager.setIO(IOType.ALL_CPU);
                manager.createSLiCinterface();
                manager.build();
        }
}
```




```
import com.maxeler.maxcompiler.v2.kernelcompiler.Kernel;

class BrightnessKernel extends Kernel {

    private static final DFEType type = dfeInt(32);

    BrightnessKernel(KernelParameters parameters) {
        super(parameters);

        Memory<DFEVar> sqrtLUT = mem.alloc(type, 256);
        DFEVar input1 = io.input("array1", type);
        DFEVar input2 = io.input("array2", type);
        DFEVar input3 = io.input("array3", type);

        sqrtLUT.mapToCPU("sqrtLut" );
        DFEVar result1 = sqrtLUT.read(input1.cast(dfeUInt(8)));
        DFEVar result2 = sqrtLUT.read(input2.cast(dfeUInt(8)));
        DFEVar result3 = sqrtLUT.read(input3.cast(dfeUInt(8)));

        io.output("outImage1", result1, type);
        io.output("outImage2", result2, type);
        io.output("outImage3", result3, type);
    }
```

HVALA NA PAŽNJI !

mn13318@alas.matf.bg.ac.rs

mr13054@alas.matf.bg.ac.rs