# Geometry
## Assignment 8

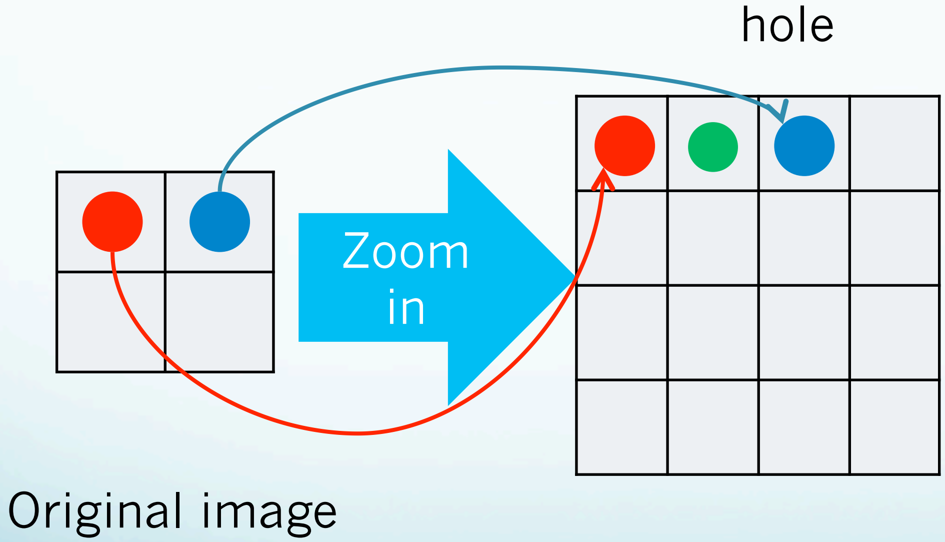
Gwenaelle Cunha Sergio
ABR Lab – BEP
KNU 2014.1

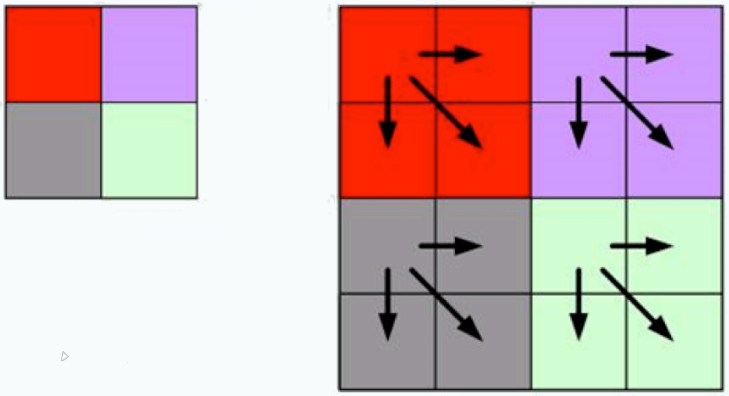
# Geometry

- Branch of mathematics concerned with the image’s shape, size, position, and all properties of space
- Transformation: rearranges pixels in the image
- Mapping: forward and backward

# Zoom-in (Nearest Neighbor)

- Replication



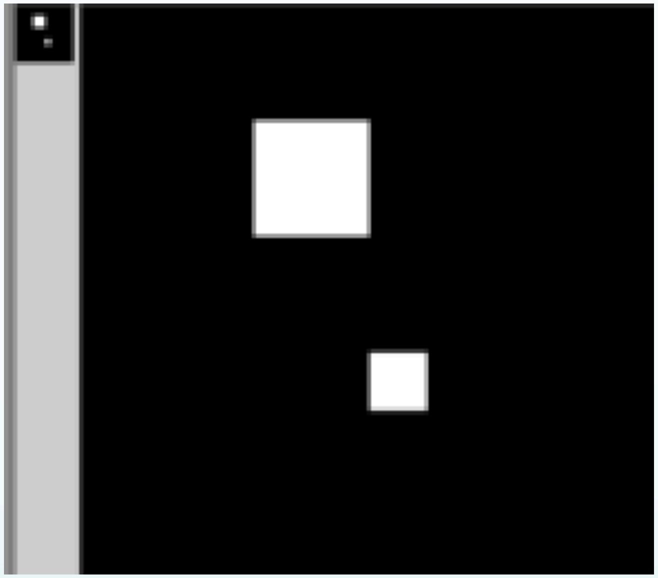


# Zoom-in (Nearest Neighbor) - Code

```
int zoomFactor = 10;
Mat img = nearestNeighborInterpolation(mat, zoomFactor);
```

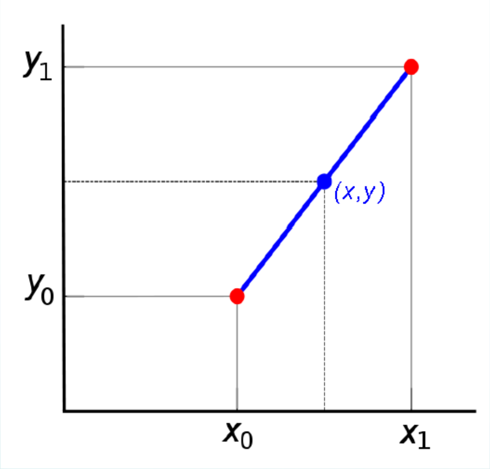
```
Mat nearestNeighborInterpolation(Mat mat, int zoomFactor) {
    cv::Mat img = cv::Mat(zoomFactor*mat.size().height, zoomFactor*mat.size().width, mat.type());
    for (int i = 0; i < mat.rows; i++){
        for (int j = 0; j < mat.cols; j++){
            for (int ki = 0; ki < zoomFactor; ki++){
                for (int kj = 0; kj < zoomFactor; kj++){
                    img.at<uchar>(ki+i*zoomFactor,kj+j*zoomFactor) = mat.at<uchar>(i,j);
                }
            }
        }
    }
    return img;
}
```

# Zoom-in (Nearest Neighbor) - Result



# Zoom-in (Linear)

- Same as before but using Linear Interpolation between 2 points



Linear slope: $\dfrac{y-y_0}{x-x_0}=\dfrac{y_1-y_2}{x_1-x_2}$

$$\Rightarrow\ y-y_0=\text{slope}\ *(x-x_0)$$

$$\Rightarrow\ y\ =\ y_0+(x-x_0)\left(\frac{y_1-y_0}{x_1-x_0}\right)$$

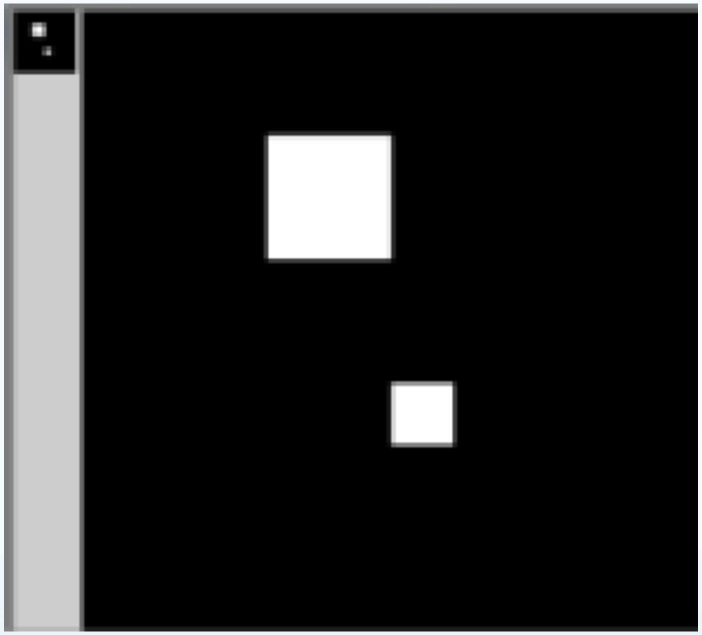
$$=\ y_0+\left(\frac{x-x_0}{x_1-x_0}\right)(y_1-y_0)$$

$$=\ \boxed{\left(1-\frac{x-x_0}{x_1-x_0}\right)}y_0+\boxed{\left(\frac{x-x_0}{x_1-x_0}\right)}y_1$$

# Zoom-in (Linear) - Code

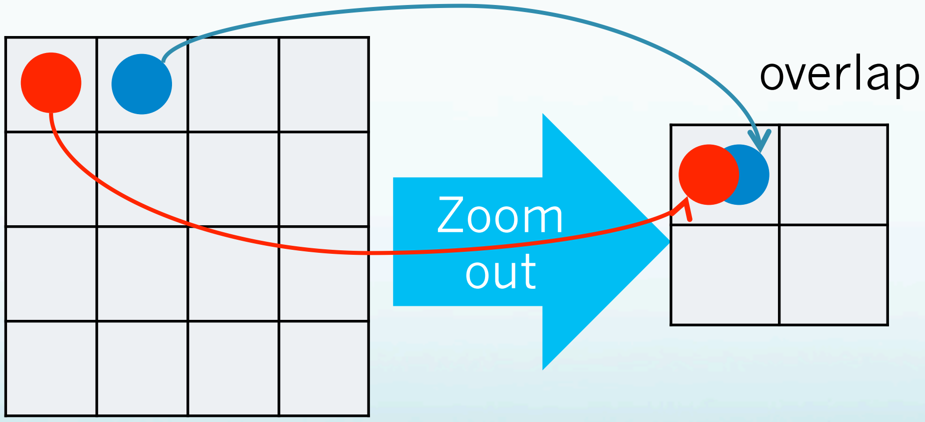
```
Mat **linearInterpolation**(Mat mat, int zoomFactor) {
    cv::Mat img = cv::Mat(zoomFactor*mat.size().height, zoomFactor*mat.size().width, mat.type());
    for (int i = 0; i < mat.rows; i++){
        for (int j = 0; j < mat.cols; j++){
            for (int ki = 0; ki < zoomFactor; ki++){
                for (int kj = 0; kj < zoomFactor; kj++){
                    int x = ki+i*zoomFactor, y = kj+j*zoomFactor;
                    int x1 = i, y1 = j, x0, y0, x0M, y0M;
                    double d, previousD;
                    for (int interpI = i-1; interpI < i+1; interpI++){ //Check border
                        for (int interpJ = j-1; interpJ < j+1; interpJ++){
                            if ((interpI >= 0) & (interpI < mat.rows) & (interpJ >= 0) & (interpJ < mat.cols)) {
                                     x0 = interpI; y0 = interpJ;
                                     d = sqrt((float) ((x1-x0)*(x1-x0) + (y1-y0)*(y1-y0)));
                                     if (interpI == i-1)  previousD = d;
                                     if (previousD > d) {
                                         previousD = d;
                                         x0M = interpI; y0M = interpJ;
                                     }
                             }
                       }
                  }
                  x0 = x0M; y0 = y0M;
                  if (x1 != x0) {
                      double grad = (y1-y0)/(x1-x0);
                      y = (int) (grad*(x-x0) + y0);
                  }
                  img.at<uchar>(x,y) = mat.at<uchar>(i,j);
                }
            }
        }
    }
    return img;
}
```

# Zoom-in (Linear) - Result



# Zoom-out (Nearest)

- Inverse of Zoom-In
- Overlap: average of points

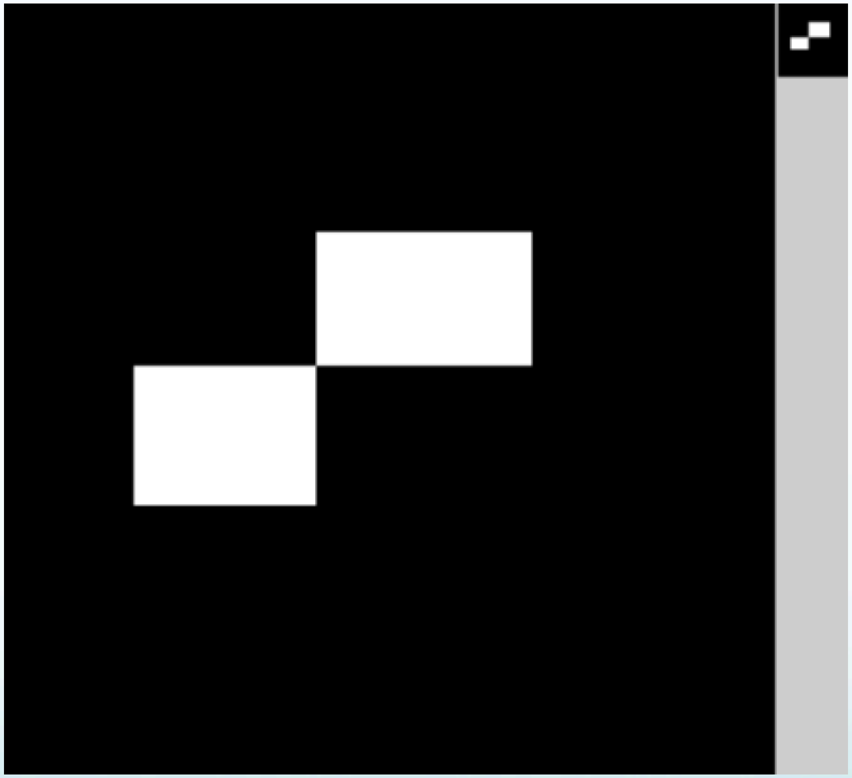


Original image

# Zoom-out (Nearest) - Code

```
int zoomFactor = 10;
Mat img = zoomOut(mat, zoomFactor);

Mat zoomOut(Mat mat, int zoomFactor) {
    cv::Mat img = cv::Mat((int) mat.size().height/zoomFactor, (int) mat.size().width/
zoomFactor, mat.type());

    for (int i = 0; i < img.rows; i++){
        for (int j = 0; j < img.cols; j++){
            for (int ki = 0; ki < zoomFactor; ki++){
                for (int kj = 0; kj < zoomFactor; kj++){
                    img.at<uchar>(i,j) = mat.at<uchar>(ki+i*zoomFactor,kj+j*zoomFactor);
                }
            }
        }
    }
    return img;
}
```

# Zoom-out (Nearest) - Result

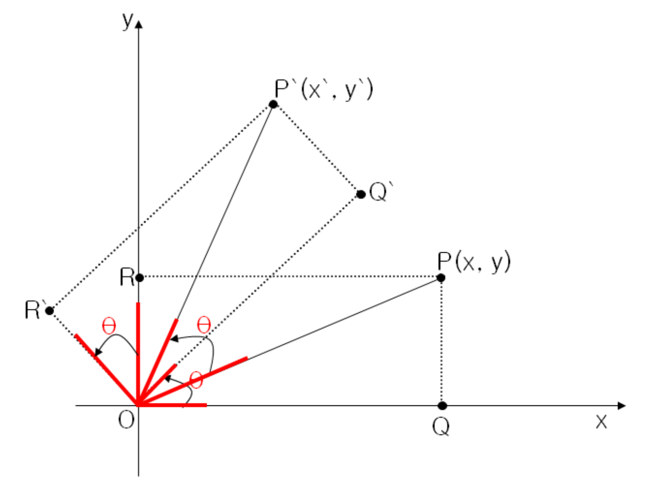


# Rotation

- Used to straighten images

- Equation:

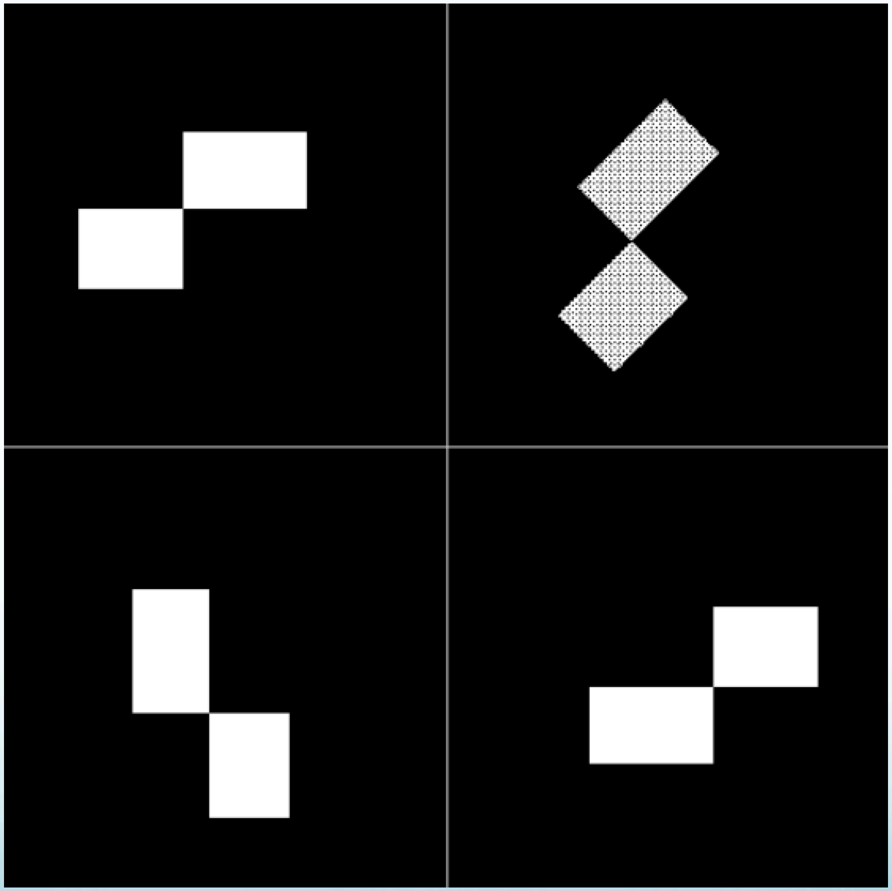
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x - C_x \\ y - C_y \end{bmatrix} + \begin{bmatrix} C_x \\ C_y \end{bmatrix}$$



# Rotation - Code

```
double theta = -PI/2;
Mat img = rotate(mat, theta);

Mat rotate(Mat mat, double theta) {
    cv::Mat img = cv::Mat(mat.size().height, mat.size().width, mat.type());
    int x0 = (int) mat.rows/2, y0 = (int) mat.cols/2;
    for (int i = 0; i < mat.rows; i++){
        for (int j = 0; j < mat.cols; j++){
             img.at<uchar>(i,j) = 0;
       }
    }
    for (int i = 0; i < mat.rows; i++){
        for (int j = 0; j < mat.cols; j++){
            int x2 = (int) (cos(theta)*(i-x0) - sin(theta)*(j-y0) + x0);
            int y2 = (int) (sin(theta)*(i-x0) + cos(theta)*(j-y0) + y0);
            if ((x2 < mat.rows) & (y2 < mat.cols)) {
                img.at<uchar>(x2,y2) = mat.at<uchar>(i,j);
            }
        }
    }
    return img;
}
```

# Rotation - Result



Original

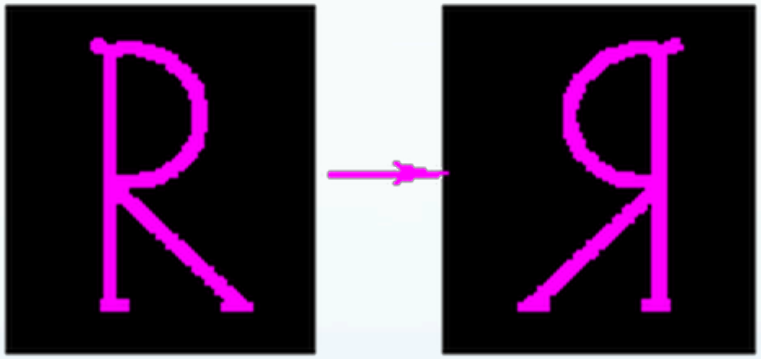
$45^{o}$

$90^{o}$

$180^{o}$

# Reflection

- Reflects pixels in the image in the x or y-axis



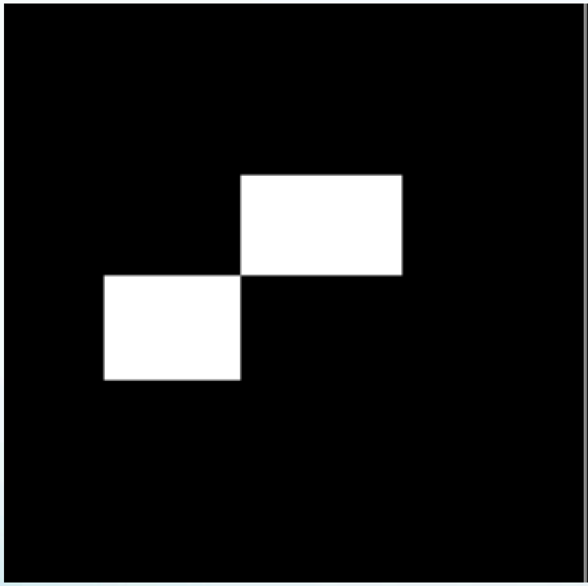
# Reflection - Code

```
Mat reflectOnX(Mat mat) {
    cv::Mat img = cv::Mat(mat.size().height, mat.size().width, mat.type());
     int x0 = (int) mat.rows/2, y0 = (int) mat.cols/2;

     for (int i = 0; i < mat.rows; i++){
         for (int j = 0; j < mat.cols; j++){
              int newX = i;
              int newY = (int) (mat.cols - j - 1);
              img.at<uchar>(newX,newY) = mat.at<uchar>(i,j);
         }
    }
    return img;
}
```

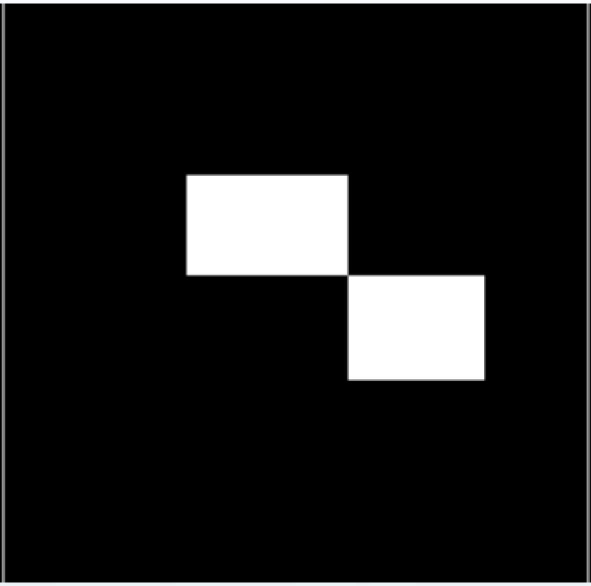
# Reflection - Code

```
Mat reflectOnY(Mat mat) {
    cv::Mat img = cv::Mat(mat.size().height, mat.size().width, mat.type());
    int x0 = (int) mat.rows/2, y0 = (int) mat.cols/2;

    for (int i = 0; i < mat.rows; i++){
        for (int j = 0; j < mat.cols; j++){
            int newX = (int) (mat.rows - i - 1);
            int newY = j;
            img.at<uchar>(newX,newY) = mat.at<uchar>(i,j);
        }
    }
    return img;
}
```

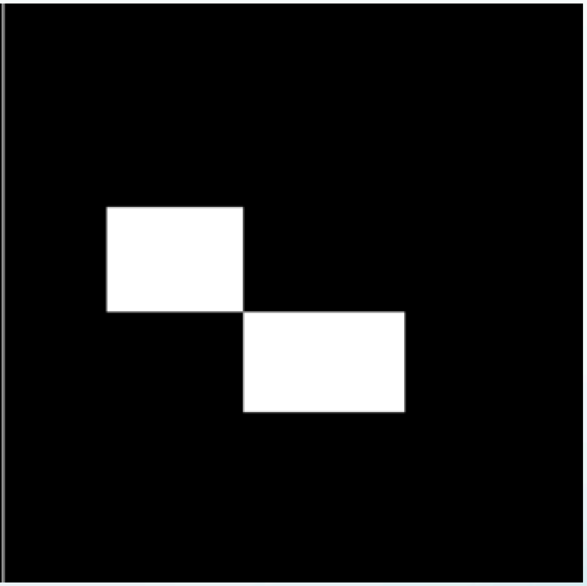
# Reflection - Result



Original image



X-axis



Y-axis