Transforms and Filters

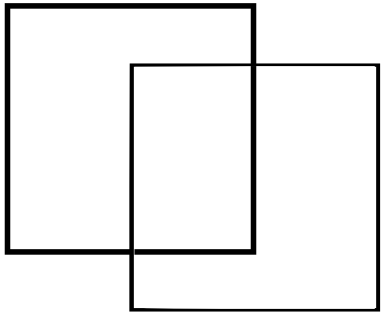
Sylvia Pan

Learning Outcomes

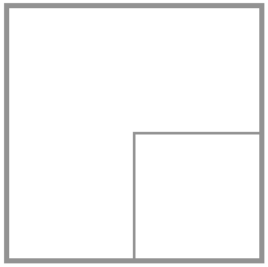
- By the end of this session you will be able
  - Use Transforms to move and manipulate graphical objects
  - Combine multiple transforms correctly
  - Explain Convolution and the how image filters work
  - Apply a built in filter to an image
  - Implement your own filters

Transforms

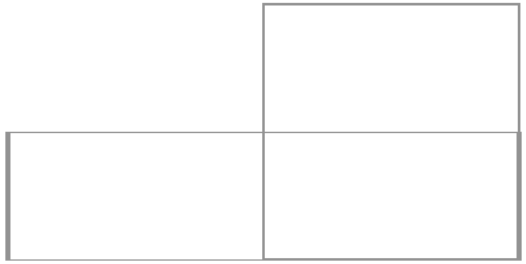
- Translate
- Rotate
- Scale



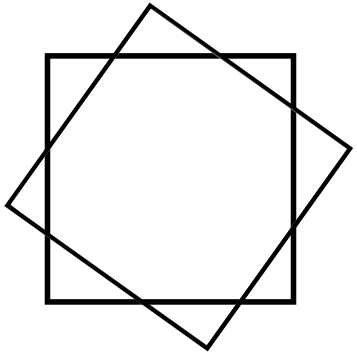
Translate



Scale

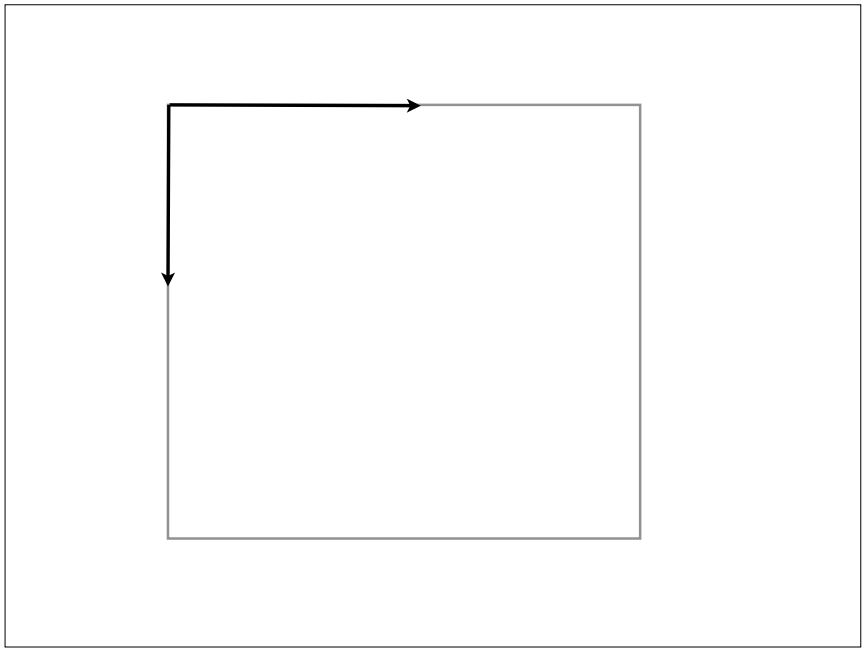


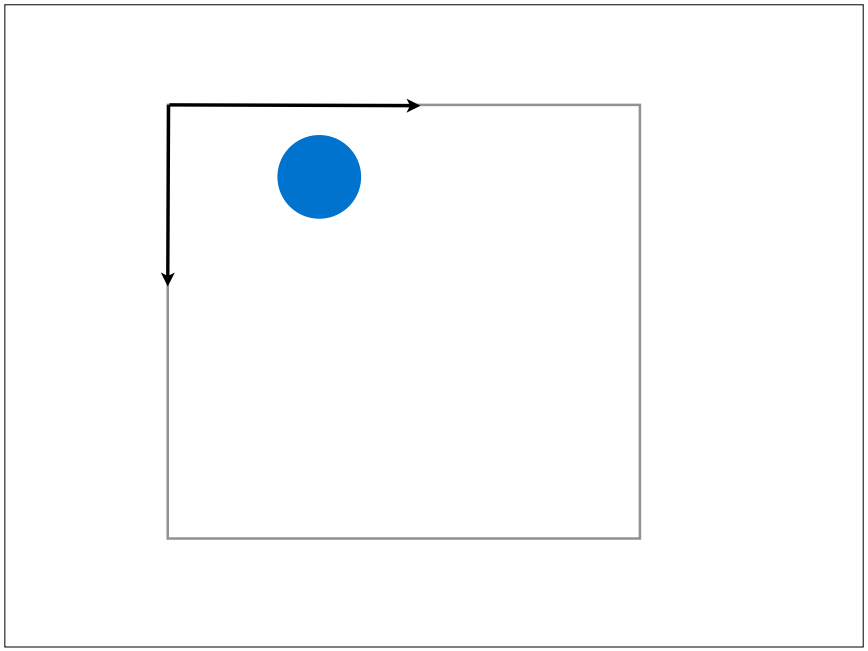
Scale

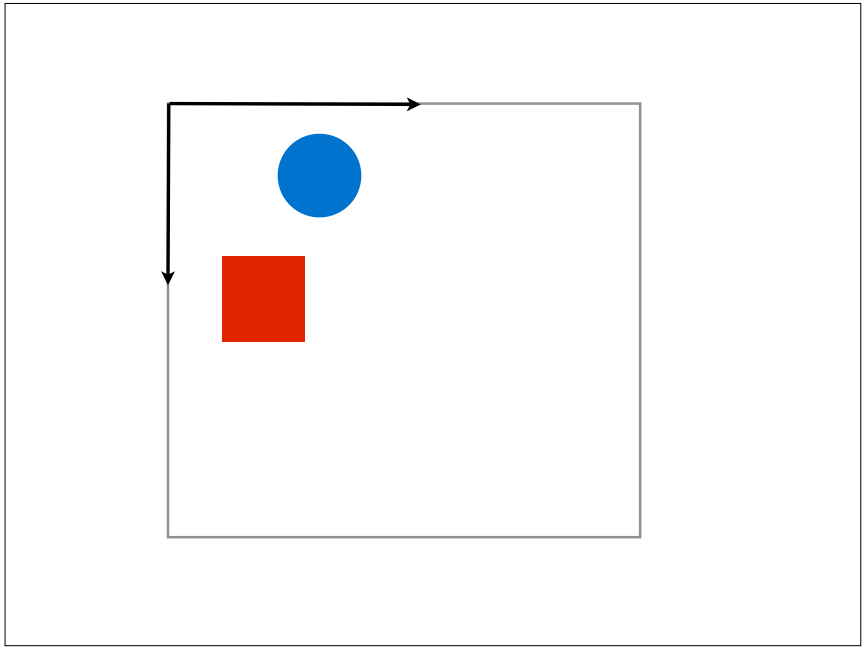


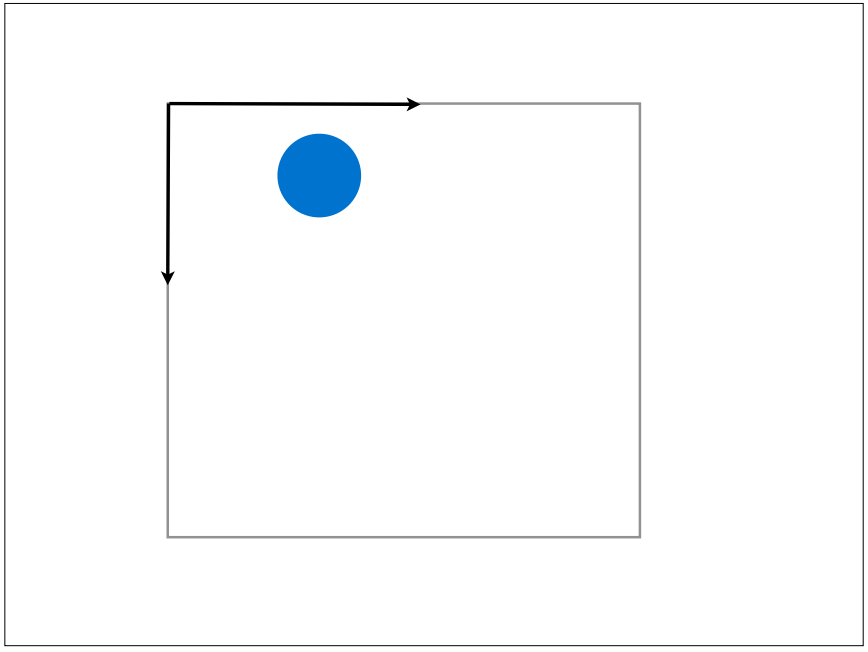
Rotate

Rotate





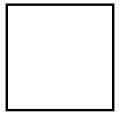




```
rect(0, 0, 200, 200);
```

A
```
translate(100, 0);
rect(0, 0, 200, 200);
```
1

B
```
rotate(radians(10));
rect(0, 0, 200, 200);
```
2



C
```
scale(2.0);
rect(0, 0, 200, 200);
```
3

D
```
scale(2.0, 1);
rect(0, 0, 200, 200);
```
4

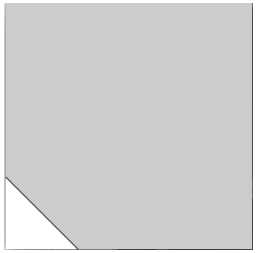
E
```
rotate(radians(45));
rect(0, 0, 200, 200);
```
5

Learning Outcomes

- By the end of this session you will be able
  - Use Transforms to move and manipulate graphical objects
  - Combine multiple transforms correctly
  - Explain Convolution and the how image filters work
  - Apply a built in filter to an image
  - Implement your own filters

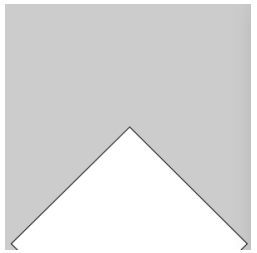
```
size(300,300);

rotate(radians(45));
translate(width/2,height/2);

rect(0,0,200,200);
```

```
size(300,300);

rotate(radians(45));
translate(width/2,height/2);

rect(0,0,200,200);
```



```
size(300,300);

translate(width/2,height/2);
rotate(radians(45));


rect(0,0,200,200);
```

```
size(300,300);

translate(width/2,height/2);
rotate(radians(45));


rect(0,0,200,200);
```



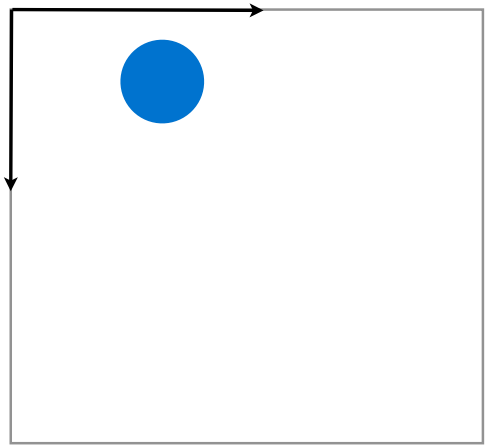
Another way to think about transformation

```
size(300,300);

rotate(radians(45));
translate(width/2,height/2);

rect(0,0,200,200);
```

2
1

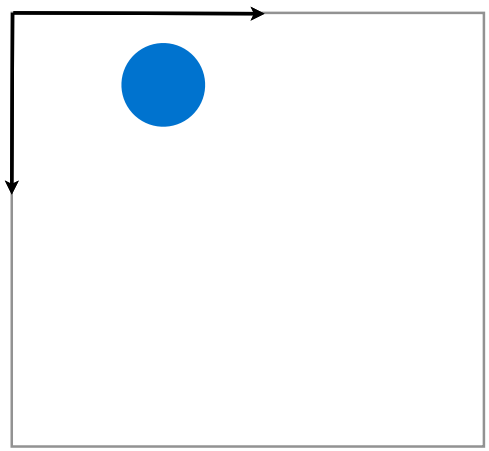
Operate on object

Original axes start fresh



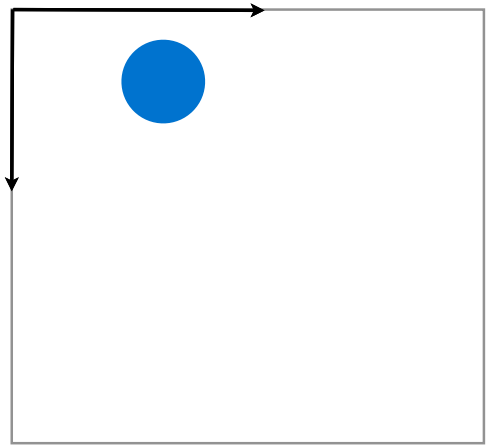
translate();
translate();

translate everything twice



translate();
rotate();

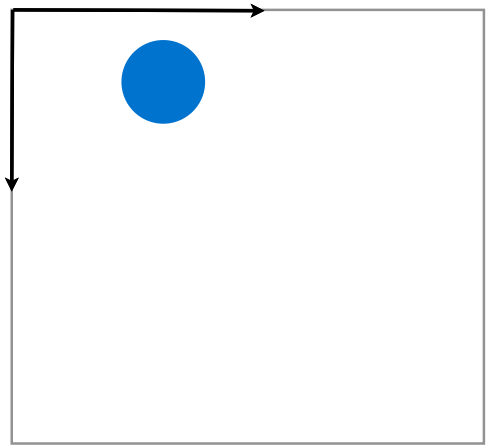
Translates the result of rotating



translate();
rotate();

rotate()
translate()

Rotate the result of translating



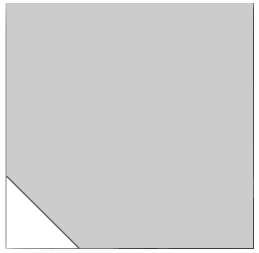
rotate()
translate()

Remember: Opposite Order

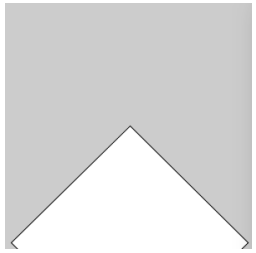
```
size(300,300);

rotate(radians(45));
translate(width/2,height/2);

rect(0,0,200,200);
```

```
size(300,300);

translate(width/2,height/2);
rotate(radians(45));

rect(0,0,200,200);
```

```
size(300,300);

rotate(radians(45));
translate(width/2,height/2);

rect(0,0,200,200);
```



```
size(300,300);

translate(width/2,height/2);
rotate(radians(45));


rect(0,0,200,200);
```

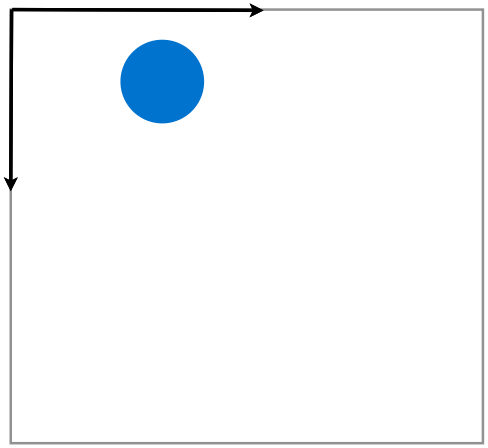


Best Order

translate();
rotate();
scale();

rotate();
translate();

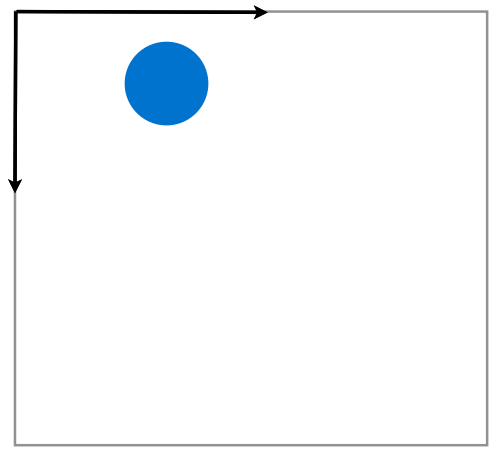
rotate();
translate();



rotate();
translate();

Doesn't end up at the end point of translating

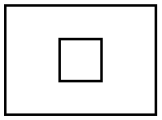
translate();
rotate();



translate();
rotate();

Rotates about its centre and moves to the correct position

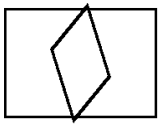
```
rect(0, 0, 50, 50);
```



A

```
rotate(1);
scale(2.0, 1);
rect(0, 0, 50, 50);
```

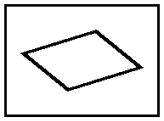
1



B

```
scale(2.0, 1);
rotate(1);
rect(0, 0, 50, 50);
```

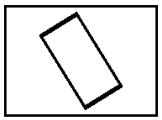
2



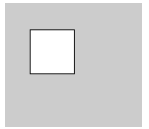
C

```
scale(1.0, 2.0);
rotate(1);
rect(0, 0, 50, 50);
```

3



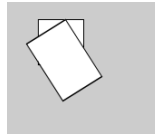
```
translate(50,50);
rect(0,0,50,50);
```



A

```
rotate(1);
translate(30, 0);
scale(1.5, 1.0);
rect(0, 0, 50, 50);
```

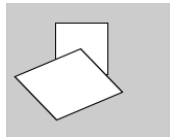
1



B

```
translate(30, 0);
rotate(1);
scale(1.5, 1.0);
rect(0, 0, 50, 50);
```

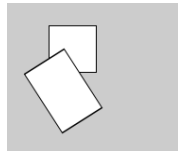
2



C

```
scale(1.5, 1.0);
rotate(1);
translate(30, 0);
rect(0, 0, 50, 50);
```

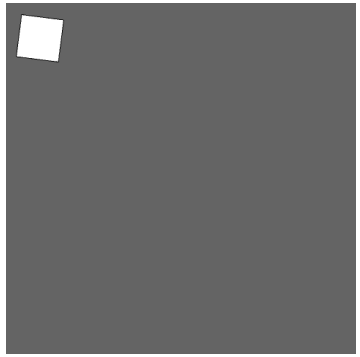
3



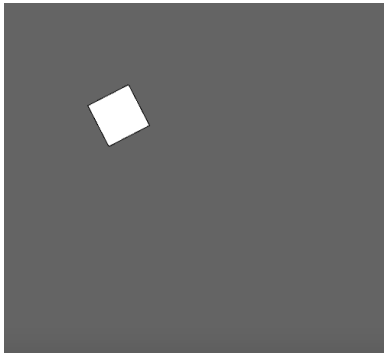
Best Order

translate();
rotate();
scale();

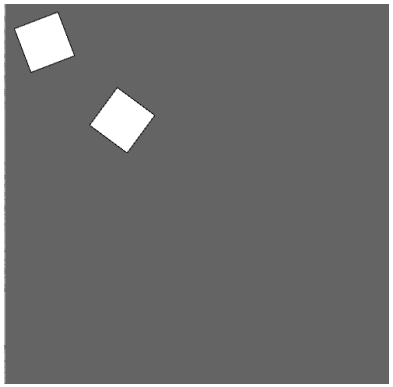
```
float theta1 = 0;
float theta2 = 1;


void setup(){
  size(500,500,P3D);
}

void draw(){
  background(100);

  rectMode(CENTER);


  translate(50,50);
  rotate(theta1);
  rect(0,0,60,60);

  theta1 += 0.02;
}
```



```
float theta1 = 0;
float theta2 = 1;


void setup(){
  size(500,500,P3D);
}

void draw(){
  background(100);

  rectMode(CENTER);

  translate(150,150);
  rotate(theta2);
  rect(0,0,60,60);

  theta2 += 0.02;
}
```



```
void draw(){
  background(100);

  rectMode(CENTER);

  pushMatrix();
  translate(50,50);
  rotate(theta1);
  rect(0,0,60,60);
  popMatrix();

  theta1 += 0.02;

  translate(150,150);
  rotate(theta2);
  rect(0,0,60,60);

  theta2 += 0.02;
}
```



Learning Outcomes

- By the end of this session you will be able
  - Use Transforms to move and manipulate graphical objects
  - Combine multiple transforms correctly
  - Explain Convolution and the how image filters work
  - Apply a built in filter to an image
  - Implement your own filters

Multiple Transforms

```
pushMatrix();
  // transform
  // draw
popMatrix();
```

Learning Outcomes

- By the end of this session you will be able
  - Use Transforms to move and manipulate graphical objects
  - Combine multiple transforms correctly
  - Explain Convolution and the how image filters work
  - Apply a built in filter to an image
  - Implement your own filters

Filters

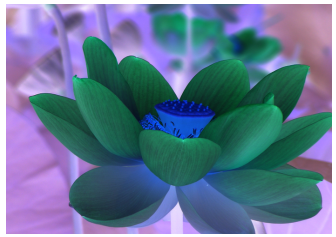
Filters

Filters are a mechanism to perform manipulations on an entire image
http://www.processing.org/reference/filter_.html
http://www.processing.org/reference/tint_.html

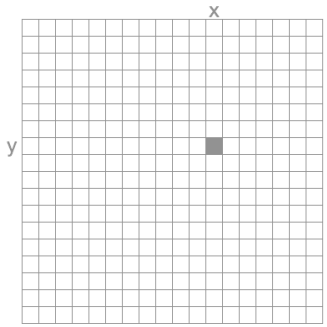
Gray scale

Threshold

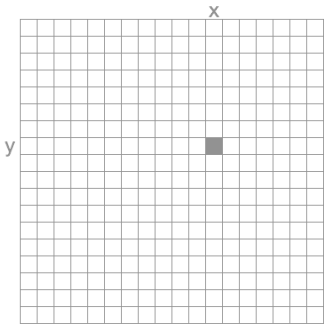
Invert



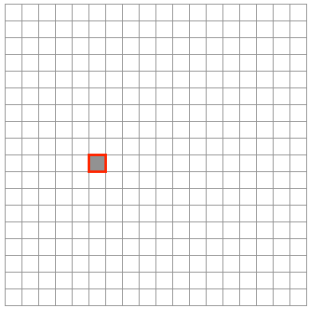
Original



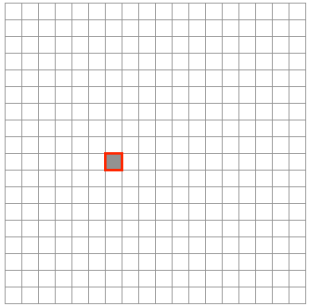
New



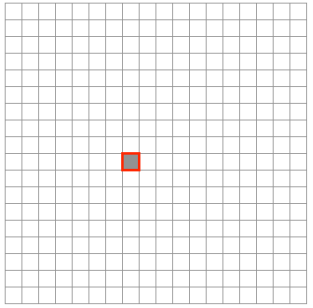
Filters under the hood



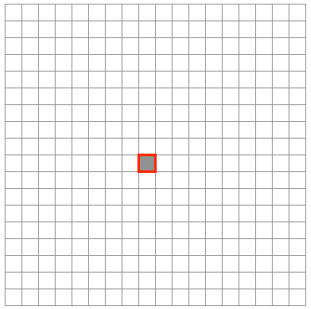
Filters under the hood



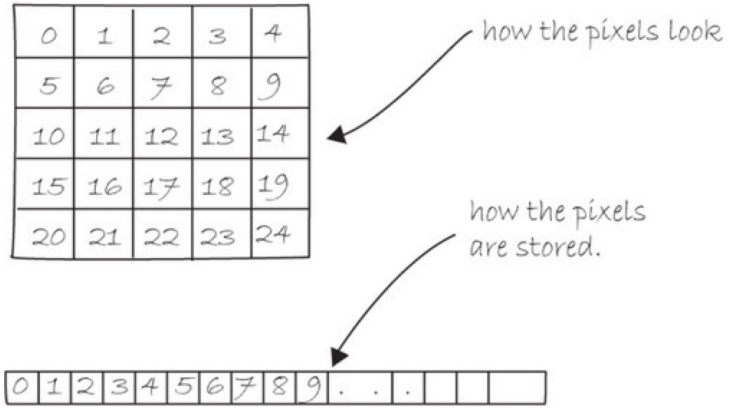
Filters under the hood



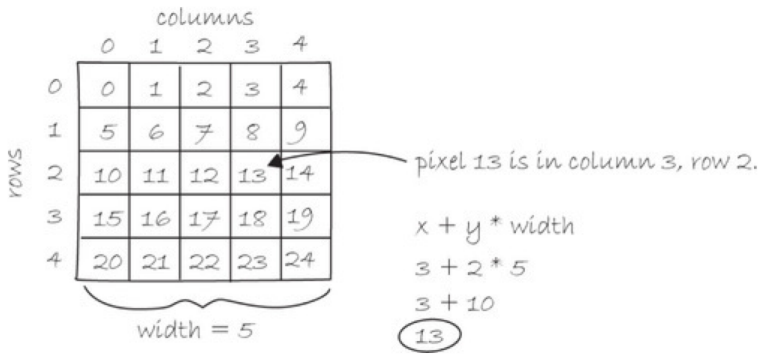
Filters under the hood



Pixels



Pixels



Which does the code do?

A

```
for (int x = 0; x < img.width; x++){
  for (int y = 0; y < img.height; y++)
  {
    img.set ( x, y, color(255) );
  }
}
```

B

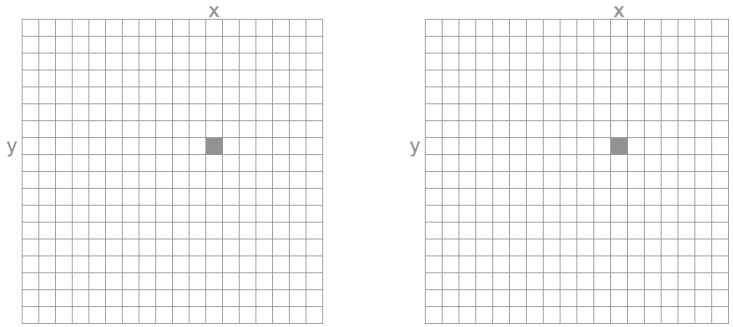
```
int imgLength = img.width * img.height;
 for(int i = 0; i < imgLength;  i++)
{
 img.pixels[i]  = color(255);
}
```

C

```
for(int x = 0; x < img.width; x++)
 {
  for(int y = 0; y < img.height; y++)
  {
   int loc = x+ y * img.width;
   img.pixels[loc] = color(255);
  }
 }
```

Original

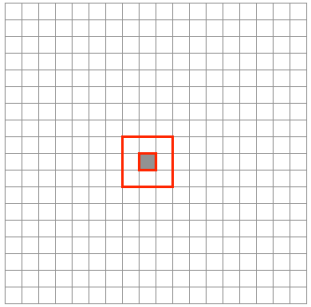
New



Filters

Filters are a mechanism to perform manipulations on an entire image
http://www.processing.org/reference/filter_.html
http://www.processing.org/reference/tint_.html

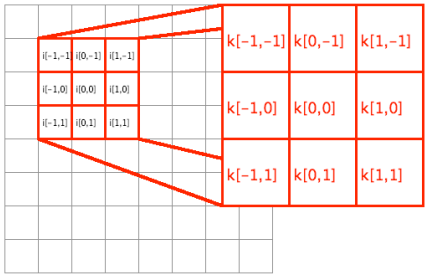
Neighbourhood filters



Filters under the hood

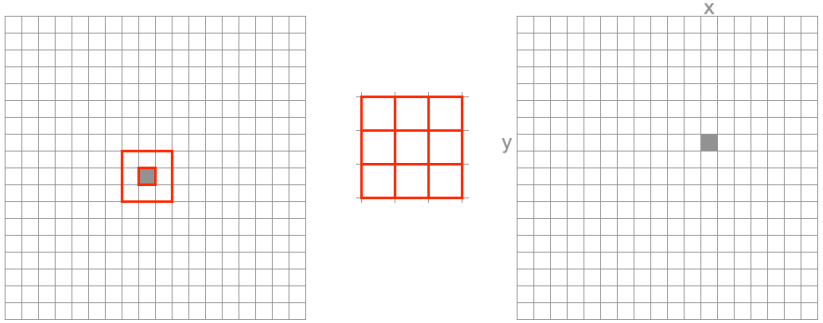
- Which of the following are neighbourhood and which are local?
  - Blur
  - Threshold
  - Gray
  - Edge detect

Kernels

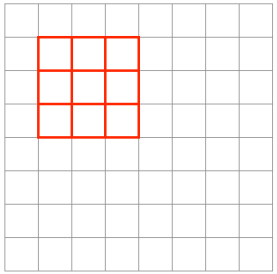


Original

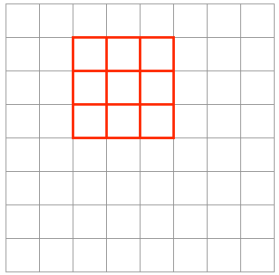
New



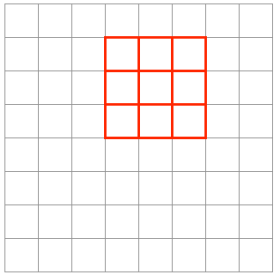
Kernels



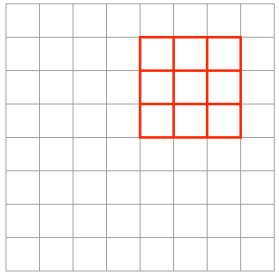
Kernels



Kernels



Kernels



Kernels

Kernels: Blur

```
float[][] kernel = { { 1.0/9, 1.0/9, 1.0/9 } ,
                     { 1.0/9, 1.0/9, 1.0/9 } ,
                     { 1.0/9, 1.0/9, 1.0/9 } } ;
```

Kernels: Sharpen

```
float[][] kernel = { { -1, -1, -1 } ,
                     { -1, 9, -1 } ,
                     { -1, -1, -1 } } ;
```

Kernels: Edge Detect

```
float[][] kernel = { { -1,  0, 1 } ,
                     { -2,  0, 2 } ,
                     { -1,  0, 1 } } ;
```

Kernels

- What do these two kernels do?

```
float[][] kernel = { { 1.0/16, 2.0/16, 1.0/16 } ,
                     { 2.0/16, 4.0/16, 2.0/16 } ,
                     { 1.0/16, 2.0/16, 1.0/16 } } ;
```

```
float[][] kernel = { { -1,  -2, -1 } ,
                     {  0,   0,  0 } ,
                     {  1,   2,  1 } } ;
```

Kernels: Gaussian Blur

```
float[][] kernel = { { 1.0/16, 2.0/16, 1.0/16 } ,
                     { 2.0/16, 4.0/16, 2.0/16 } ,
                     { 1.0/16, 2.0/16, 1.0/16 } } ;
```

Kernels: Edge Detect (Vertical)

```
float[][] kernel = { { -1,  -2, -1 } ,
                     {  0,   0,  0 } ,
                     {  1,   2,  1 } } ;
```

Kernels - some concepts

- Sum of Kernel values:
  - The sum of a blur kernel is normally 1 to preserve the brightness from the original image.
  - Edge detect filter normally sums to 0 and produces a darker image
- Low pass and high pass filter
  - Low pass filter filters out high frequency, and produce more smooth looking images. For instance, blur.
  - High pass filter filters out low are

Learning Outcomes

- By the end of this session you will be able
  - Use Transforms to move and manipulate graphical objects
  - Combine multiple transforms correctly
  - Explain Convolution and the how image filters work
  - Apply a built in filter to an image
  - Implement your own filters