# OpenCL exercise 3: Sobel filter

Kaicong Sun

# Sobel filter

- Used for edge detection in images
- A combination of two convolutional operators: horizontal and vertical

$$G_1 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \qquad G_2 = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A$$

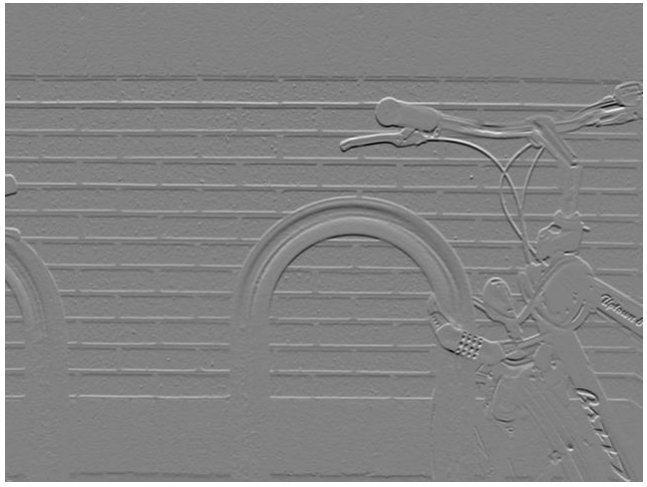
$$G = \sqrt{G_1^2 + G_2^2}$$

# Sobel filter

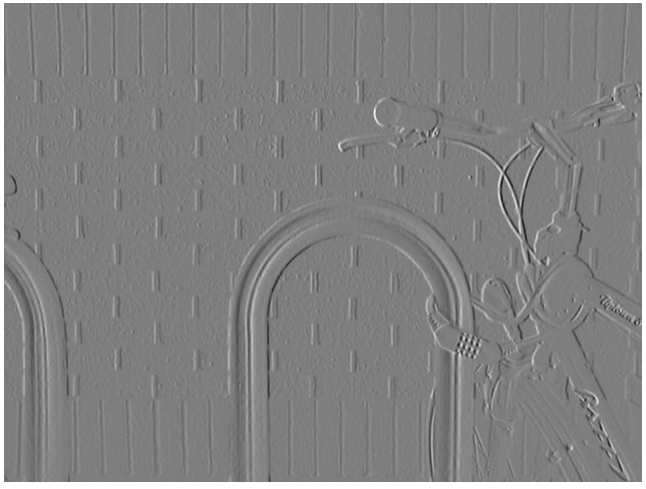
```
1  G1(i,j)=
2        1*A(i-1, j-1) +2*A(i-1, j  ) +1*A(i-1, j+1)
3      +         0       +      0          +       0
4      -1*A(i+1, j-1) -2*A(i+1, j  ) -1*A(i+1, j+1)
5
6  G2(i,j)=
7        1*A(i-1, j-1) +         0       -1*A(i-1, j+1)
8      +2*A(i   , j-1) +         0       -2*A(i   , j+1)
9      +1*A(i+1, j-1) +         0       -1*A(i+1, j+1)
10
11 G(i,j)=sqrt(G1(i,j)*G1(i,j)+ G2(i,j)*G2(i,j))
```

# Sobel filter

Original image



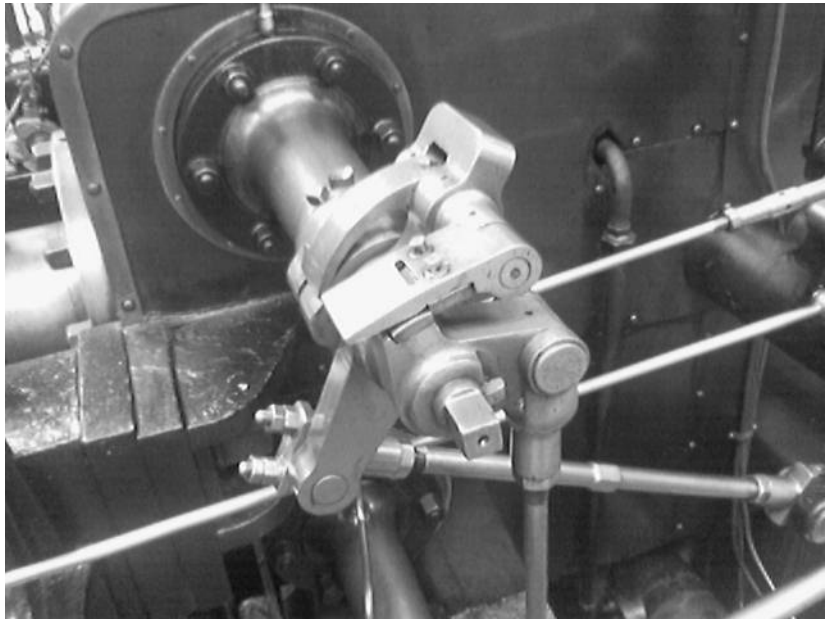
$G_1$



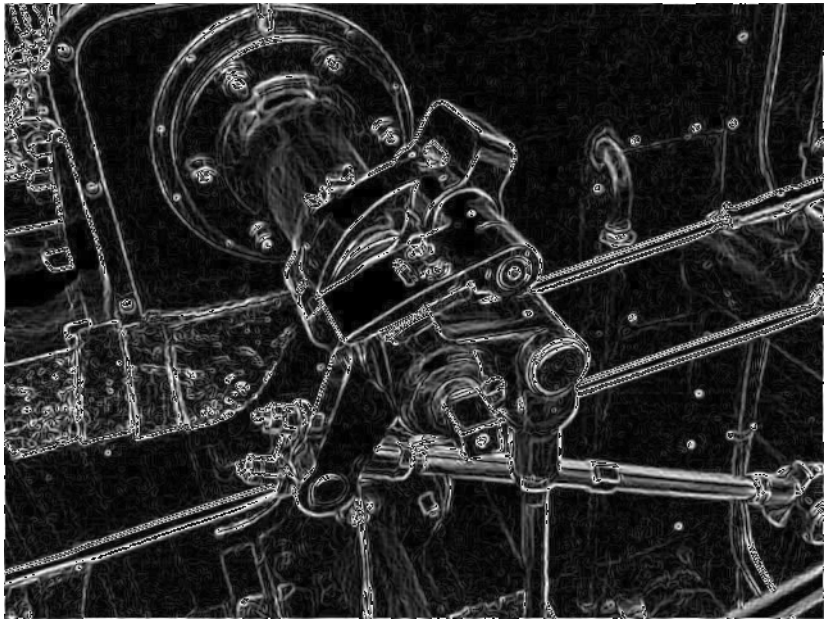
$G_2$

Output image

# Sobel filter



Original image



Output of Sobel filter

# Task 1

- Implement the sobel filter on the GPU, similar to the CPU implementation (using global memory)
- Write profiling code: Speedup and MPixel/s

# Task 2

- Make a copy of the kernel created in Task 1 and modify it to make sure that the four corner pixels are only loaded once
- Compare the performance to Task 1

# Task 3

- Make a copy of the kernel created in Task 2 and use an OpenCL image for the input data
- Compare the performance to Task 1 and Task 2

# OpenCL Images

- Same as CUDA “Texture Memory”
- Is a 1D / 2D / 3D array on the GPU
- Can be accessed using “samplers”
- Provide caching
  - Accesses via spatial coordinates (x,y)
- Additional features
  - Coordinate normalization
    - x/y/z coordinates go from 0.0 to 1.0
  - Return special value for out-of-bounds access
  - Filtering (i.e. linear/bilinear/trilinear interpolation)

# OpenCL Images / Samplers

Samplers can be used to access an OpenCL Image on the GPU.
Sampler Options:

- Coordinate normalization:
  - `CLK_NORMALIZED_COORDS_FALSE`: Coordinates to from 0 to width-1/height-1
  - `CLK_NORMALIZED_COORDS_TRUE`: Coordinates to from 0 to 1
- Addressing mode: (for out-of-bounds accesses)
  - `CLK_ADDRESS_NONE`: Undefined behavior
  - `CLK_ADDRESS_CLAMP`: Return 0
  - `CLK_ADDRESS_CLAMP_TO_EDGE`: Return color of border
  - `CLK_ADDRESS_REPEAT`: Repeat image
  - `CLK_ADDRESS_MIRRORED_REPEAT`: Repeat mirrored image
- Filtering:
  - `CLK_FILTER_NEAREST`: Nearest neighbor
  - `CLK_FILTER_LINEAR`: Linear/Bilinear/Trilinear interpolation

# OpenCL Images / Syntax Host

Creating an Image:

```
cl::Image2D::Image2D(cl::Context context,
    cl_mem_flags flags, cl::ImageFormat format,
    std::size_t width, std::size_t height);
```

`context` = The OpenCL context to use
`flags` = Normally `CL_MEM_READ_ONLY`
`format` = The content of the image, e.g. `cl::ImageFormat(CL_R, CL_FLOAT)`, `CL_R` = 1 channel, `CL_FLOAT` = contains floats
`width` = Width of the image
`height` = Height of the image

# OpenCL Images / Syntax Host

Copying data to an image:

```
cl::CommandQueue::enqueueWriteImage(cl::Image& image,
  cl_bool blocking,
  cl::size_t<3> origin, cl::size_t<3> region,
  std::size_t row_pitch, std::size_t slice_pitch, void* ptr,
  eventsToWaitFor = NULL, cl::Event* event = NULL) const;
```

`image` = The destination image
`blocking` = Wait until the copy operation has finished (normally `true`)
`origin` = The origin of the destination region (see next slide)
`region` = The size of the destination region (see next slide)
`row_pitch` = Number of bytes between two rows, normally `width * sizeof(ElementType)`
`slice_pitch` = Bytes between two slices, for 2D images use `0`
`ptr` = Pointer to source data

# OpenCL Images / Syntax Host

Syntax for `cl::size_t<3>`:

```
cl::size_t<3> origin;
origin[0] = origin[1] = origin[2] = 0;

cl::size_t<3> region;
region[0] = width;
region[1] = height;
region[2] = 1;
queue.enqueueWriteImage(..., origin, region, ...);

Kernel.setArg<cl::Image2D>(0, image);
```

`origin` = The origin of the destination region, normally (0, 0, 0)
`region` = The size of the destination region, for 2D images normally (width, height, 1)

# OpenCL Images / Syntax Kernel

Syntax for using an image:

```
// declare sampler
const sampler_t sampler = CLK_NORMALIZED_COORDS_TRUE
    | CLK_ADDRESS_REPEAT | CLK_FILTER_LINEAR;

// pass image as parameter
__kernel void sobelKernel(__read_only image2d_t d_input, ...)

// read value at (i, j)
float f = read_imagef(d_input, sampler, (int2){i, j}).x;
```