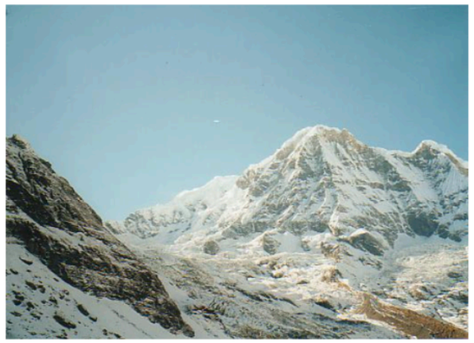
ECE277 Final Project

SIFT Feature Matching

Chi-Hsin Lo 20220314

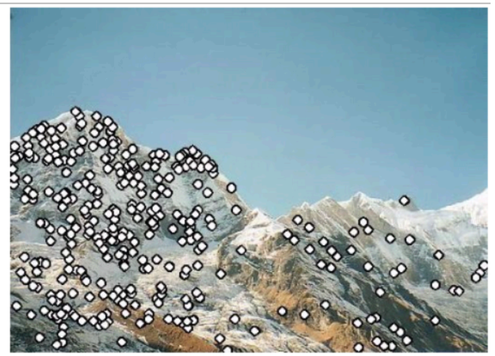
SIFT (Scale Invariant Feature Transform)



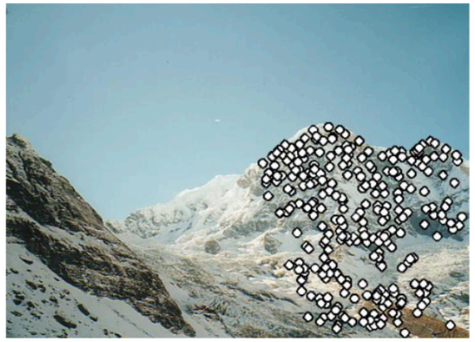
(a) Image 1

(b) Image 2

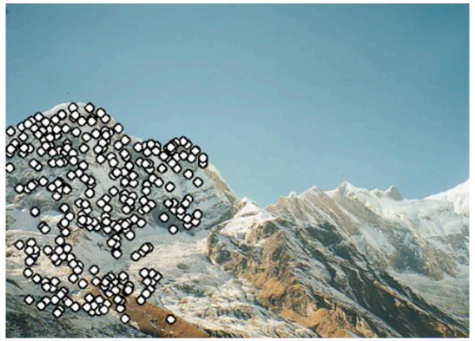
(c) SIFT matches 1



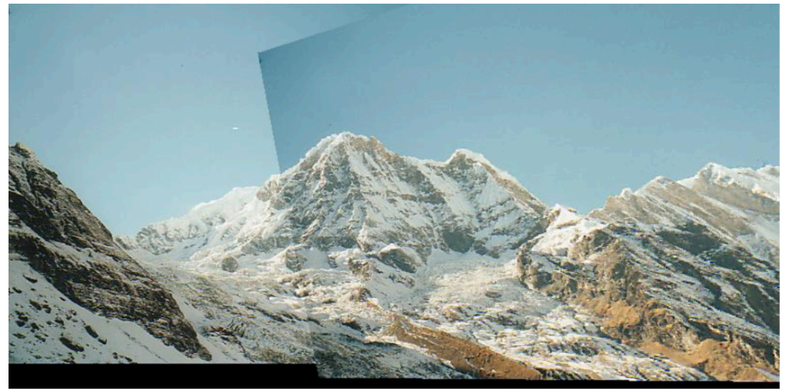
(d) SIFT matches 2



(e) RANSAC inliers 1



(f) RANSAC inliers 2

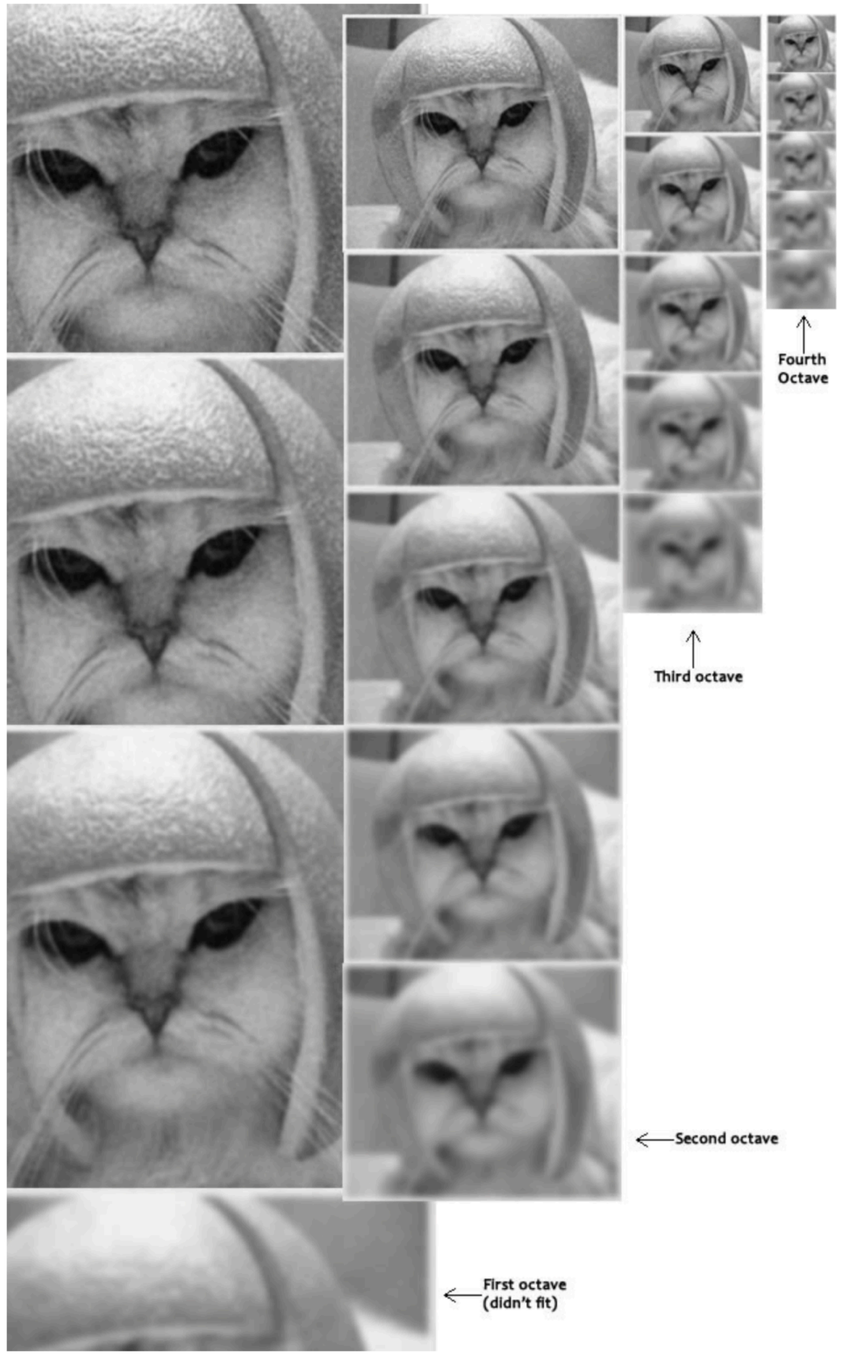


(g) Images aligned according to a homography

SIFT Overview

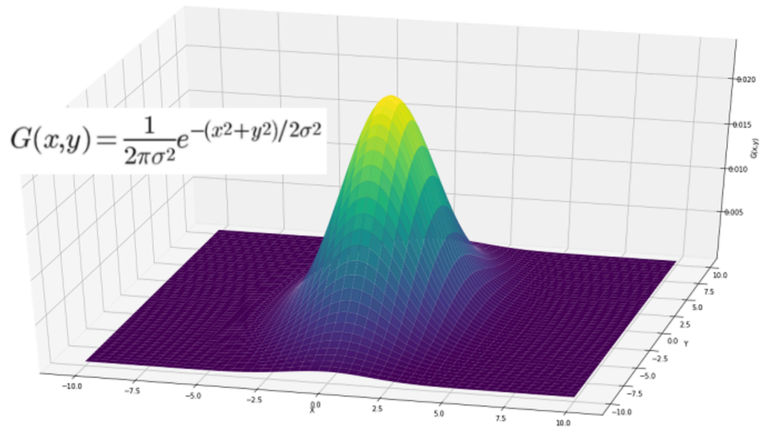
Scale Space

- Real world objects are meaningful only at a certain scale.
- You might see a sugar cube perfectly on a table. But if looking at the entire milky way, then it simply does not exist.
- 4 octaves and 5 blur levels



SIFT Overview

Gaussian Blur



$$G(x,y,\sigma)=\frac{1}{2\pi\sigma^2}e^{-(x^2+y^2)/2\sigma^2}$$

Gaussian Blur operator

$$L(x,y,\sigma)=G(x,y,\sigma)*I(x,y)$$

Blurred image

- L : blurred image
- G : Gaussian Blur operator
- I : image
- x, y : location coordinates

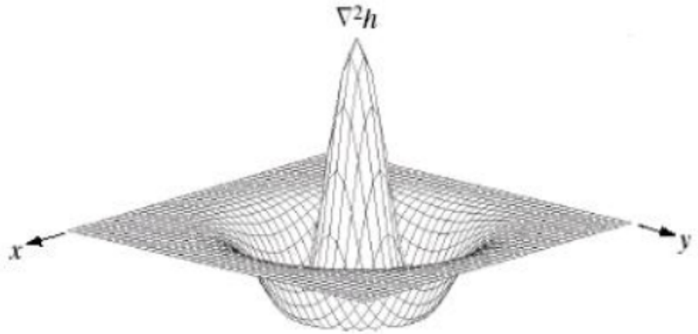
octave \ scale →				
0.707107	1.000000	1.414214	2.000000	2.828427
1.414214	2.000000	2.828427	4.000000	5.656854
2.828427	4.000000	5.656854	8.000000	11.313708
5.656854	8.000000	11.313708	16.000000	22.627417

σ : amount of blur

- σ : ”Scale" parameter, amount of blur. Greater the value, greater the blur.
- * : convolution operation. It "applies" gaussian blur G onto the image I.

SIFT Overview

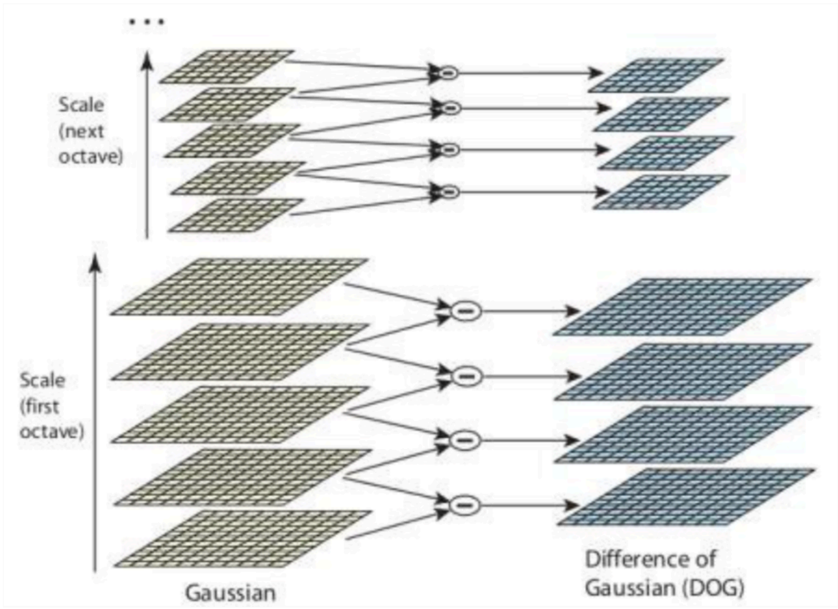
$$\nabla \cdot \nabla f = \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$



- Laplacian of Gaussian $\nabla^2 G$
- Scale Invariant Laplacian of Gaussian $\sigma^2 \nabla^2 G$
- Locates edges and corners on the image
- Edges and corners are good for finding keypoints
- These Difference of Gaussian images

  are approximately equivalent to the

  Laplacian of Gaussian.

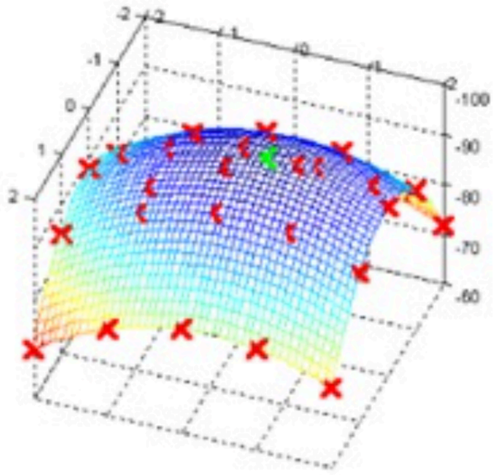


Gaussian Pyramid

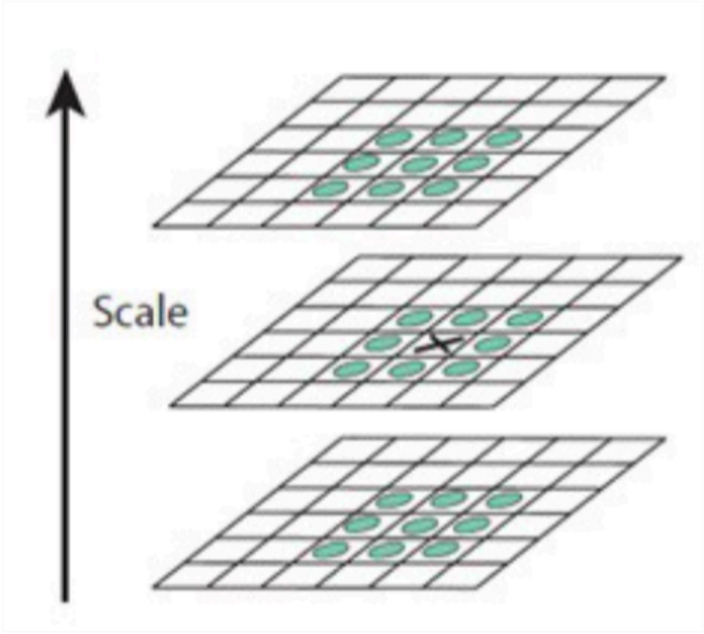
SIFT Overview

Finding Key Points

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}$$



- Iterate through each pixel and check all its neighbours
- X is marked as a "key point" if it is the greatest or least of all 26 neighbours
- No iteration on lowermost and topmost scales, there simply aren't enough neighbours



SIFT Overview

Remove Low Contrast Key Points

- Remove low contrast key points :

  Reject if the magnitude < a certain value

- The image around a key point can be :

  A Flat region : both gradients are small

  An Edge : 1 big (perpendicular to edge) 1 small (along the edge)

  A Corner : both gradients are big

  Pass both gradients are big enough (we only want corner), by Hessian Matrix

- Reject flats:
  - $|D(\hat{\mathbf{x}})|$ < 0.03
- Reject edges:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Let α be the eigenvalue with larger magnitude and β the smaller.

$$\mathrm{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta,$$

$$\mathrm{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta.$$

Let r = α/β. So α = rβ

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r},$$

$(r+1)^2/r$ is at a min when the 2 eigenvalues are equal.

  - r < 10

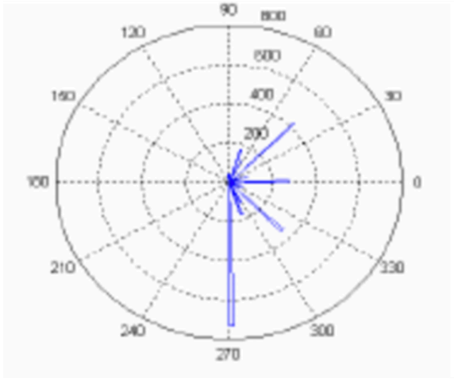
SIFT Overview

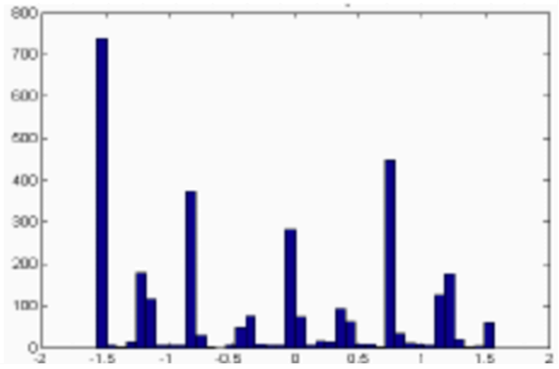
Magnitude and Orientation

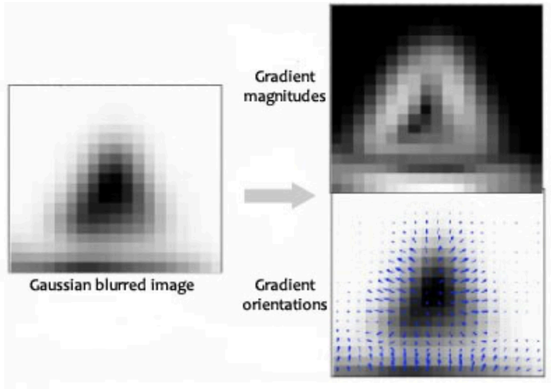
- Calculate magnitude and orientation for all pixels around the key point

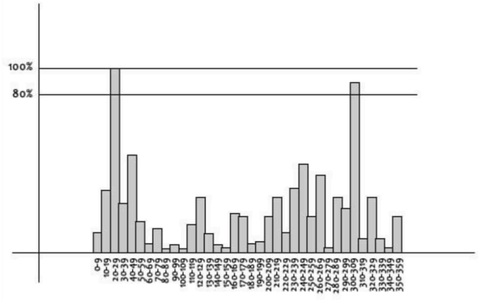
$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$



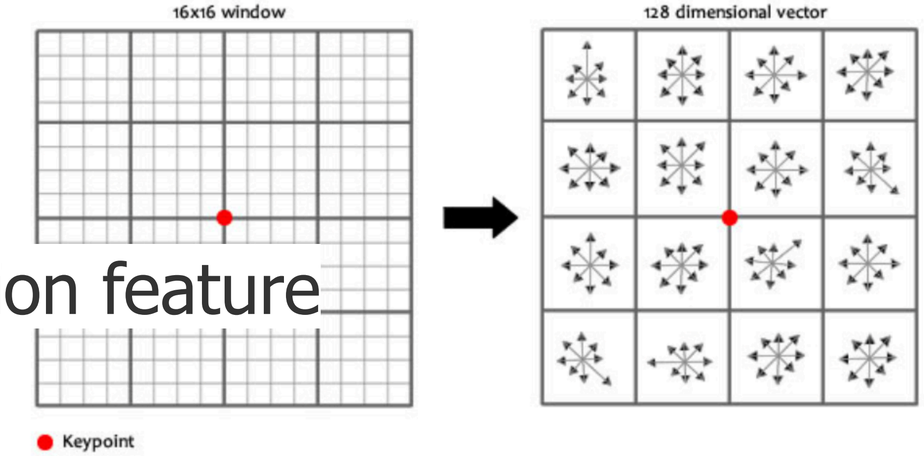






SIFT Overview

Generating Feature

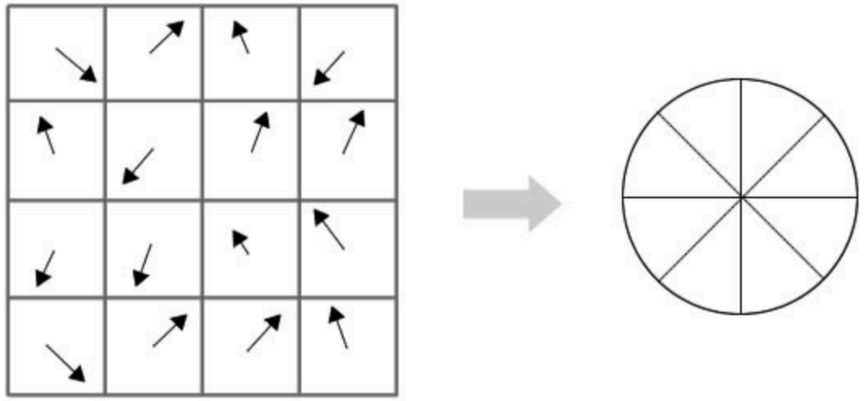


4x4x8 = 128 dimension feature

- Acquire a 16x16 window around the keypoint
- Break it to sixteen 4x4 windows
- Calculate gradient magnitudes and orientations

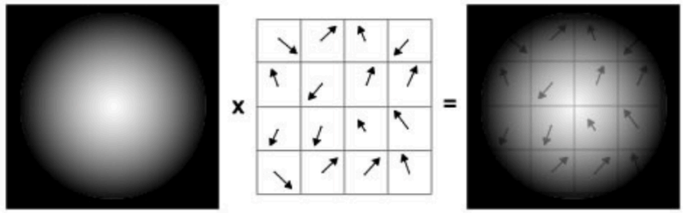
  in each 4x4 window, put orientations into an

  8 bin histogram (45 degree each bin)



- Amount added depends on the magnitude of the gradient

  and distance to key point, by Gaussian weighting function



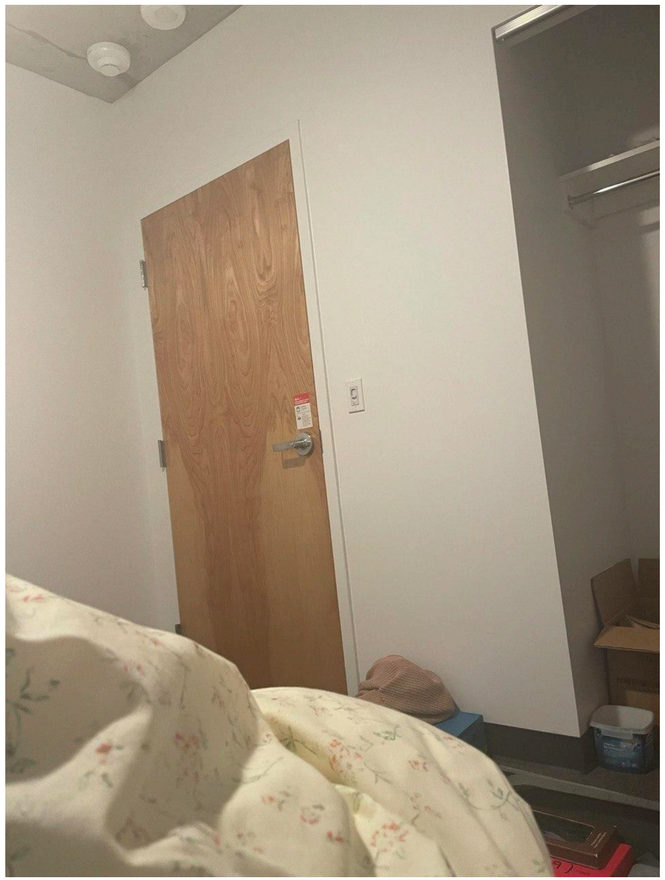
Project Overview

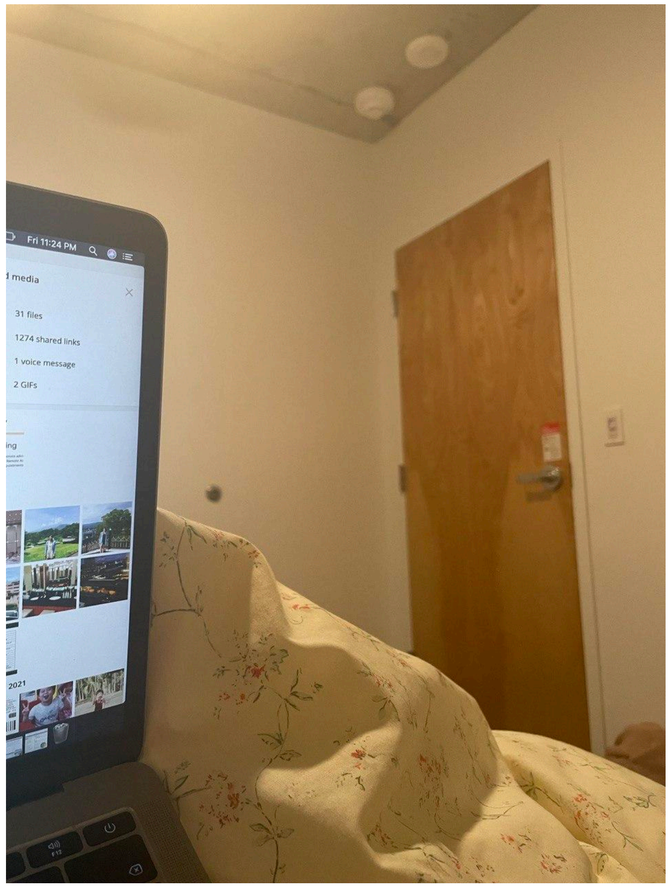
Overview

- (0) Read Image
- (1) Compute 20 Gaussian Kernels (4 octaves, 5 blur levels)
- (2) Compute 4 Padded Octaves
- (3) Compute 20 Gaussian Blurs (4 octaves, 5 blur levels)
- (4) Compute 16 DOGs (4 octaves, 4 DOGs)
- (5) Compute Key Points
- ~~(6) Compute Orientation and Magnitude~~
- ~~(7) Compute Feature for Key Points~~
- ~~(8) Match Key Points~~

Project Overview

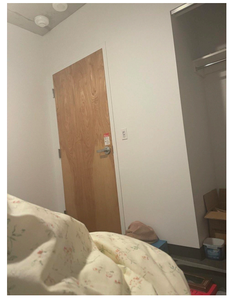
(0) img1 and img2

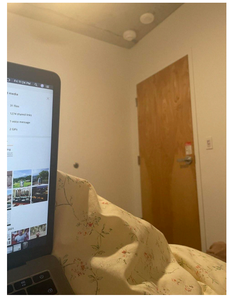




Project Overview

(0) Read Image - main





float* image1

nrows1 * ncols1

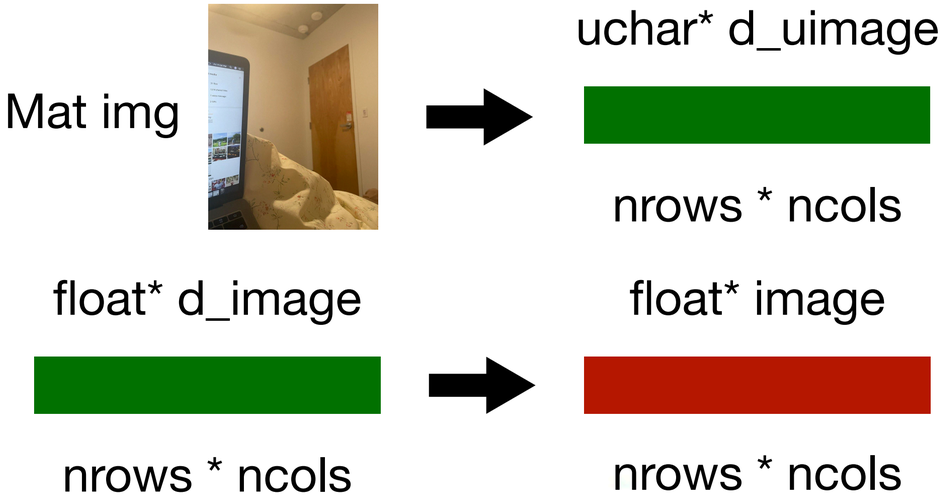
float* image2

nrows2 * ncols2

```
//0. opencv read image and put into cpu float array
// opencv read image
string img1_path = "C:/Users/c2lo/Desktop/img1.jpg", img2_path = "C:/Users/c2lo/Desktop/img2.jpg";
Mat img1 = imread(img1_path.c_str(), IMREAD_GRAYSCALE);
Mat img2 = imread(img2_path.c_str(), IMREAD_GRAYSCALE);
const unsigned int nrows1 = img1.rows, ncols1 = img1.cols, nrows2 = img2.rows, ncols2 = img2.cols;
float *image1 = (float*)malloc(nrows1 * ncols1 * sizeof(float));
float *image2 = (float*)malloc(nrows2 * ncols2 * sizeof(float));
opencvImageToCpuFloat(img1, image1); opencvImageToCpuFloat(img2, image2);
//printCvDataAndImageFloatArray(img1, image1); printCvDataAndImageFloatArray(img2, image2);
```

Project Overview

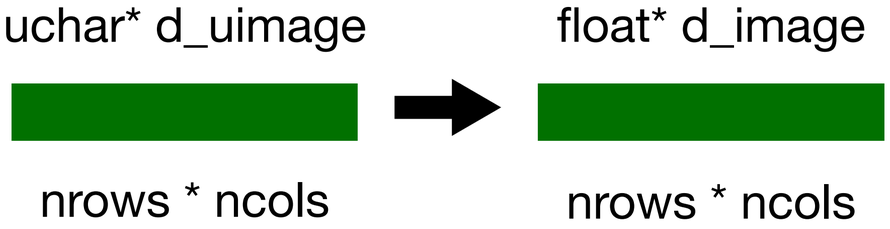
(0) Read Image - host function

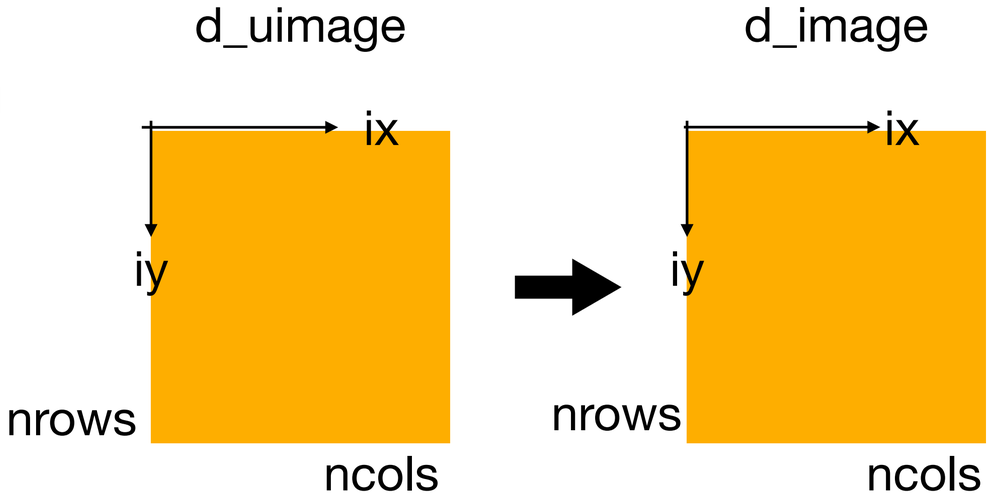



```
void opencvImageToCpuFloat(Mat & img, float *image) {
    // read cv Mat cpu uchar array to gpu uchar array
    unsigned char *d_uimage;
    const int nrows = img.rows, ncols = img.cols;
    cudaMalloc((void**)&d_uimage, nrows * ncols * sizeof(unsigned char));
    cudaMemcpy(d_uimage, img.data, nrows * ncols * sizeof(unsigned char), cudaMemcpyHostToDevice);

    // gpu uchar array to gpu float array
    float *d_image;
    cudaMalloc((void**)&d_image, nrows * ncols * sizeof(float));

    int nx = ncols, ny = nrows, dimx = 32, dimy = 32;
    dim3 block(dimx, dimy);
    dim3 grid((nx + block.x - 1) / block.x, (ny + block.y - 1) / block.y);
    gpuUcharToFloat << <grid, block >> > (d_image, d_uimage, nx);

    // gpu float array to cpu float array
    cudaMemcpy(image, d_image, nrows * ncols * sizeof(float), cudaMemcpyDeviceToHost);

    // free and cuda free
    cudaFree(d_uimage); cudaFree(d_image);
}
```

Project Overview

(0) Read Image - device function






```
__global__ void gpuUcharToFloat(float *d_a, unsigned char *d_b, const unsigned int nx) {
    const unsigned int ix = blockIdx.x * blockDim.x + threadIdx.x;
    const unsigned int iy = blockIdx.y * blockDim.y + threadIdx.y;
    const unsigned int i = iy * nx + ix; // read and write by row
    d_a[i] = (float)d_b[i];
}
```

Project Overview

(1) Compute 20 Gaussian Kernels (4 octaves, 5 blur levels) - main

float* kernel

octave_num * kernel_num * kernel_size * kernel_size

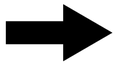
```
//1. compute kernels
// compute gpu kernel float array
const unsigned int  octave_num = 4, kernel_num = 5, kernel_size = 5; //must be odd
float *kernel = (float*)malloc(octave_num* kernel_num * kernel_size * kernel_size * sizeof(float));
cpuComputeKernel(kernel, octave_num, kernel_num, kernel_size);
//printKernel(kernel, octave_num, kernel_num, kernel_size);
```

Project Overview

(1) Compute 20 Gaussian Kernels (4 octaves, 5 blur levels) - host function

float* d_kernel

octave_num * kernel_num * kernel_size * kernel_size



float* kernel

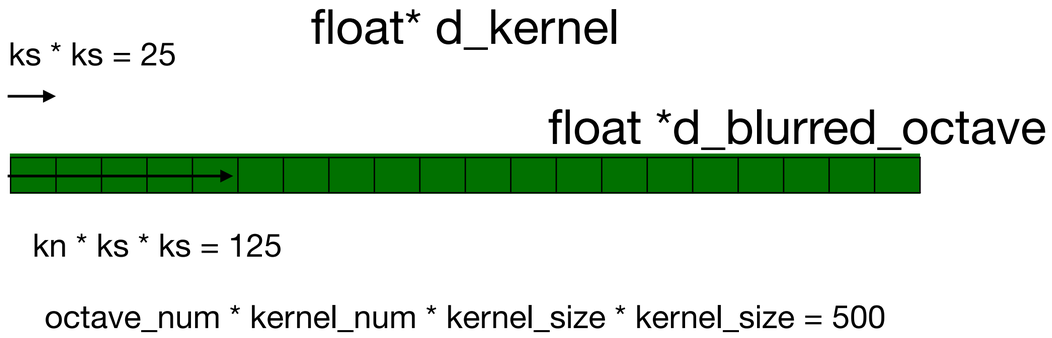
octave_num * kernel_num * kernel_size * kernel_size

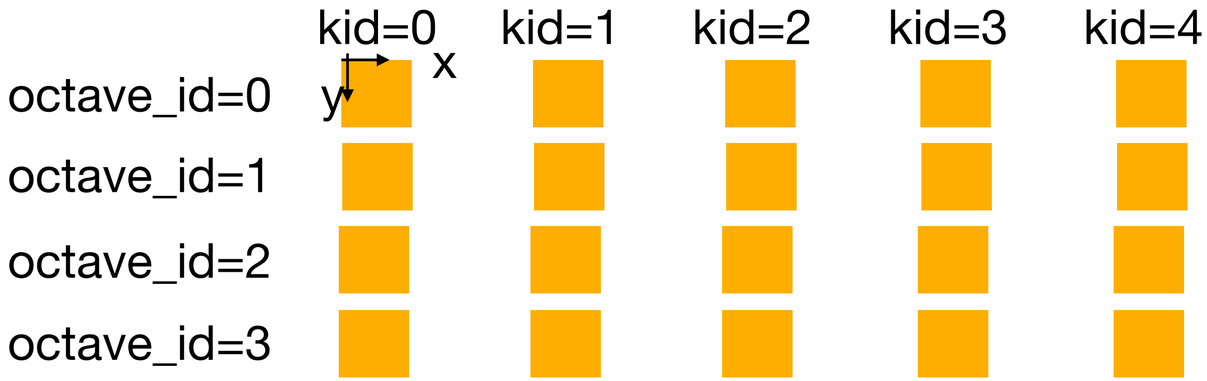
```
void cpuComputeKernel(float *kernel, const unsigned int on, const unsigned int kn, const unsigned int ks) {
    // cuda<alloc gpu float array
    float* d_kernel;
    cudaMalloc((void**)&d_kernel, on * kn * ks * ks * sizeof(float));

    // compute gpu float array
    const unsigned int nx = 32, dimx = 32, dimy = 16;
    dim3 block(dimx, dimy);
    dim3 grid(1, 1);

    gpuComputeKernel << <grid, block >> > (d_kernel, on, kn, ks, nx);

    // cudaMemcpy gpu float array to cpu float array
    cudaMemcpy(kernel, d_kernel, on * kn * ks * ks * sizeof(float), cudaMemcpyDeviceToHost);

    // free and cudaFree
    cudaFree(d_kernel);
}
```

Project Overview

octave \ scale				
0.707107	1.000000	1.414214	2.000000	2.828427
1.414214	2.000000	2.828427	4.000000	5.656854
2.828427	4.000000	5.656854	8.000000	11.313708
5.656854	8.000000	11.313708	16.000000	22.627417

(1) Compute 20 Gaussian Kernels (4 octaves, 5 blur levels) - device function






```
__global__ void gpuComputeKernel(float *d_a, const unsigned int on, const unsigned int kn, const unsigned int ks, const unsigned int nx) {
    const unsigned int ix = blockIdx.x * blockDim.x + threadIdx.x;
    const unsigned int iy = blockIdx.y * blockDim.y + threadIdx.y;
    const unsigned int i = iy * nx + ix; // read and write by row
    if (i >= 500) return;
    // i/25 is kernel id, i%5 -2 is kernel.x ([-2,-1,0,1,2]), (i%25)/5 -2 is kernel.y ([-2,-1,0,1,2])
    const unsigned int octave_id = i / 125, kernel_id = (i % 125) / 25;
    const unsigned int x = i % 5 - 2, y = (i % 25) / 5 - 2;
    const float sigma_square = powf(2, (kernel_id + 2.0 * octave_id - 1.0) / 2);
    const float power = (x * x + y * y) / (-2 * sigma_square);
    const float denominator = 2 * PI * sigma_square;
    d_a[i] = expf(power) / denominator;
    //printf("i = %d, ix = %d, iy = %d, octave_id = %d, kernel_id = %d, x = %d, y = %d, sigma_square = %f, power = %f, denominator = %f, valu
}
```

ncols * scale

Project Overview

(1) Compute 20 Gaussian Kernels (4 octaves, 5 blur levels) - result

```
octave_id = 0, kernel_id = 0
0.000786 0.006560 0.013303 0.006560 0.000786
0.006560 0.054720 0.110979 0.054720 0.006560
0.013303 0.110979 0.225079 0.110979 0.013303
0.006560 0.054720 0.110979 0.054720 0.006560
0.000786 0.006560 0.013303 0.006560 0.000786
--------------------------------
octave_id = 0, kernel_id = 1
0.002915 0.013064 0.021539 0.013064 0.002915
0.013064 0.058550 0.096532 0.058550 0.013064
0.021539 0.096532 0.159155 0.096532 0.021539
0.013064 0.058550 0.096532 0.058550 0.013064
0.002915 0.013064 0.021539 0.013064 0.002915
--------------------------------
octave_id = 0, kernel_id = 2
0.006652 0.019212 0.027360 0.019212 0.006652
0.019212 0.055490 0.079024 0.055490 0.019212
0.027360 0.079024 0.112540 0.079024 0.027360
0.019212 0.055490 0.079024 0.055490 0.019212
0.006652 0.019212 0.027360 0.019212 0.006652
--------------------------------
octave_id = 0, kernel_id = 3
0.010770 0.022799 0.029275 0.022799 0.010770
0.022799 0.048266 0.061975 0.048266 0.022799
0.029275 0.061975 0.079577 0.061975 0.029275
0.022799 0.048266 0.061975 0.048266 0.022799
0.010770 0.022799 0.029275 0.022799 0.010770
--------------------------------
octave_id = 0, kernel_id = 4
0.013680 0.023249 0.027745 0.023249 0.013680
0.023249 0.039512 0.047152 0.039512 0.023249
0.027745 0.047152 0.056270 0.047152 0.027745
0.023249 0.039512 0.047152 0.039512 0.023249
0.013680 0.023249 0.027745 0.023249 0.013680
--------------------------------
```

```
octave_id = 1, kernel_id = 0
0.006652 0.019212 0.027360 0.019212 0.006652
0.019212 0.055490 0.079024 0.055490 0.019212
0.027360 0.079024 0.112540 0.079024 0.027360
0.019212 0.055490 0.079024 0.055490 0.019212
0.006652 0.019212 0.027360 0.019212 0.006652
--------------------------------
octave_id = 1, kernel_id = 1
0.010770 0.022799 0.029275 0.022799 0.010770
0.022799 0.048266 0.061975 0.048266 0.022799
0.029275 0.061975 0.079577 0.061975 0.029275
0.022799 0.048266 0.061975 0.048266 0.022799
0.010770 0.022799 0.029275 0.022799 0.010770
--------------------------------
octave_id = 1, kernel_id = 2
0.013680 0.023249 0.027745 0.023249 0.013680
0.023249 0.039512 0.047152 0.039512 0.023249
0.027745 0.047152 0.056270 0.047152 0.027745
0.023249 0.039512 0.047152 0.039512 0.023249
0.013680 0.023249 0.027745 0.023249 0.013680
--------------------------------
octave_id = 1, kernel_id = 3
0.014637 0.021297 0.024133 0.021297 0.014637
0.021297 0.030987 0.035113 0.030987 0.021297
0.024133 0.035113 0.039789 0.035113 0.024133
0.021297 0.030987 0.035113 0.030987 0.021297
0.014637 0.021297 0.024133 0.021297 0.014637
--------------------------------
octave_id = 1, kernel_id = 4
0.013872 0.018085 0.019756 0.018085 0.013872
0.018085 0.023576 0.025755 0.023576 0.018085
0.019756 0.025755 0.028135 0.025755 0.019756
0.018085 0.023576 0.025755 0.023576 0.018085
0.013872 0.018085 0.019756 0.018085 0.013872
--------------------------------
```

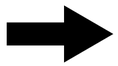
```
octave_id = 2, kernel_id = 0
0.013680 0.023249 0.027745 0.023249 0.013680
0.023249 0.039512 0.047152 0.039512 0.023249
0.027745 0.047152 0.056270 0.047152 0.027745
0.023249 0.039512 0.047152 0.039512 0.023249
0.013680 0.023249 0.027745 0.023249 0.013680
--------------------------------
octave_id = 2, kernel_id = 1
0.014637 0.021297 0.024133 0.021297 0.014637
0.021297 0.030987 0.035113 0.030987 0.021297
0.024133 0.035113 0.039789 0.035113 0.024133
0.021297 0.030987 0.035113 0.030987 0.021297
0.014637 0.021297 0.024133 0.021297 0.014637
--------------------------------
octave_id = 2, kernel_id = 2
0.013872 0.018085 0.019756 0.018085 0.013872
0.018085 0.023576 0.025755 0.023576 0.018085
0.019756 0.025755 0.028135 0.025755 0.019756
0.018085 0.023576 0.025755 0.023576 0.018085
0.013872 0.018085 0.019756 0.018085 0.013872
--------------------------------
octave_id = 2, kernel_id = 3
0.012067 0.014555 0.015494 0.014555 0.012067
0.014555 0.017557 0.018689 0.017557 0.014555
0.015494 0.018689 0.019894 0.018689 0.015494
0.014555 0.017557 0.018689 0.017557 0.014555
0.012067 0.014555 0.015494 0.014555 0.012067
--------------------------------
octave_id = 2, kernel_id = 4
0.009878 0.011278 0.011788 0.011278 0.009878
0.011278 0.012877 0.013459 0.012877 0.011278
0.011788 0.013459 0.014067 0.013459 0.011788
0.011278 0.012877 0.013459 0.012877 0.011278
0.009878 0.011278 0.011788 0.011278 0.009878
--------------------------------
```

```
octave_id = 3, kernel_id = 0
0.013872 0.018085 0.019756 0.018085 0.013872
0.018085 0.023576 0.025755 0.023576 0.018085
0.019756 0.025755 0.028135 0.025755 0.019756
0.018085 0.023576 0.025755 0.023576 0.018085
0.013872 0.018085 0.019756 0.018085 0.013872
--------------------------------
octave_id = 3, kernel_id = 1
0.012067 0.014555 0.015494 0.014555 0.012067
0.014555 0.017557 0.018689 0.017557 0.014555
0.015494 0.018689 0.019894 0.018689 0.015494
0.014555 0.017557 0.018689 0.017557 0.014555
0.012067 0.014555 0.015494 0.014555 0.012067
--------------------------------
octave_id = 3, kernel_id = 2
0.009878 0.011278 0.011788 0.011278 0.009878
0.011278 0.012877 0.013459 0.012877 0.011278
0.011788 0.013459 0.014067 0.013459 0.011788
0.011278 0.012877 0.013459 0.012877 0.011278
0.009878 0.011278 0.011788 0.011278 0.009878
--------------------------------
octave_id = 3, kernel_id = 3
0.007747 0.008508 0.008778 0.008508 0.007747
0.008508 0.009345 0.009641 0.009345 0.008508
0.008778 0.009641 0.009947 0.009641 0.008778
0.008508 0.009345 0.009641 0.009345 0.008508
0.007747 0.008508 0.008778 0.008508 0.007747
--------------------------------
octave_id = 3, kernel_id = 4
0.005894 0.006298 0.006439 0.006298 0.005894
0.006298 0.006730 0.006880 0.006730 0.006298
0.006439 0.006880 0.007034 0.006880 0.006439
0.006298 0.006730 0.006880 0.006730 0.006298
0.005894 0.006298 0.006439 0.006298 0.005894
--------------------------------
```

Project Overview

float* image1

nrows1 * ncols1



float* padded_octave1_1

(nrows1 * 2 + 2 * pad_size) * (ncols1 * 2 + 2 * pad_size)

float* padded_octave2_1

(nrows1 + 2 * pad_size) * (ncols1 + 2 * pad_size)

float* padded_octave3_1

(nrows1 * 0.5 + 2 * pad_size) * (ncols1 * 0.5 + 2 * pad_size)

float* padded_octave4_1

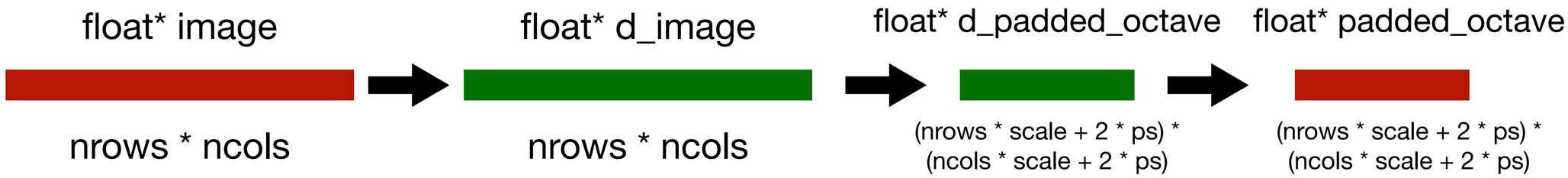
(nrows1 * 0.25 + 2 * pad_size) * (ncols1 * 0.25 + 2 * pad_size)

(2) Compute 4 Padded Octaves - main

```
//2. image to padded octaves
const unsigned int pad_size = kernel_size / 2;
const unsigned int nrows_padded_octave1_1 = nrows1 * 2 + 2 * pad_size, ncols_padded_octave1_1 = ncols1 * 2 + 2 * pad_size;
const unsigned int nrows_padded_octave2_1 = nrows1 + 2 * pad_size, ncols_padded_octave2_1 = ncols1 + 2 * pad_size;
const unsigned int nrows_padded_octave3_1 = nrows1 / 2 + 2 * pad_size, ncols_padded_octave3_1 = ncols1 / 2 + 2 * pad_size;
const unsigned int nrows_padded_octave4_1 = nrows1 / 4 + 2 * pad_size, ncols_padded_octave4_1 = ncols1 / 4 + 2 * pad_size;
float *padded_octave1_1 = (float*)malloc(nrows_padded_octave1_1 * ncols_padded_octave1_1 * sizeof(float));
float *padded_octave2_1 = (float*)malloc(nrows_padded_octave2_1 * ncols_padded_octave2_1 * sizeof(float));
float *padded_octave3_1 = (float*)malloc(nrows_padded_octave3_1 * ncols_padded_octave3_1 * sizeof(float));
float *padded_octave4_1 = (float*)malloc(nrows_padded_octave4_1 * ncols_padded_octave4_1 * sizeof(float));
//printPaddedOctavesSize(nrows1, ncols1, pad_size);
computeAndPadOctave(padded_octave1_1, image1, nrows1, ncols1, pad_size, 2.0);
computeAndPadOctave(padded_octave2_1, image1, nrows1, ncols1, pad_size, 1.0);
computeAndPadOctave(padded_octave3_1, image1, nrows1, ncols1, pad_size, 0.5);
computeAndPadOctave(padded_octave4_1, image1, nrows1, ncols1, pad_size, 0.25);
savePaddedOctaves(padded_octave1_1, padded_octave2_1, padded_octave3_1, padded_octave4_1, nrows1, ncols1, pad_size, 1);
```

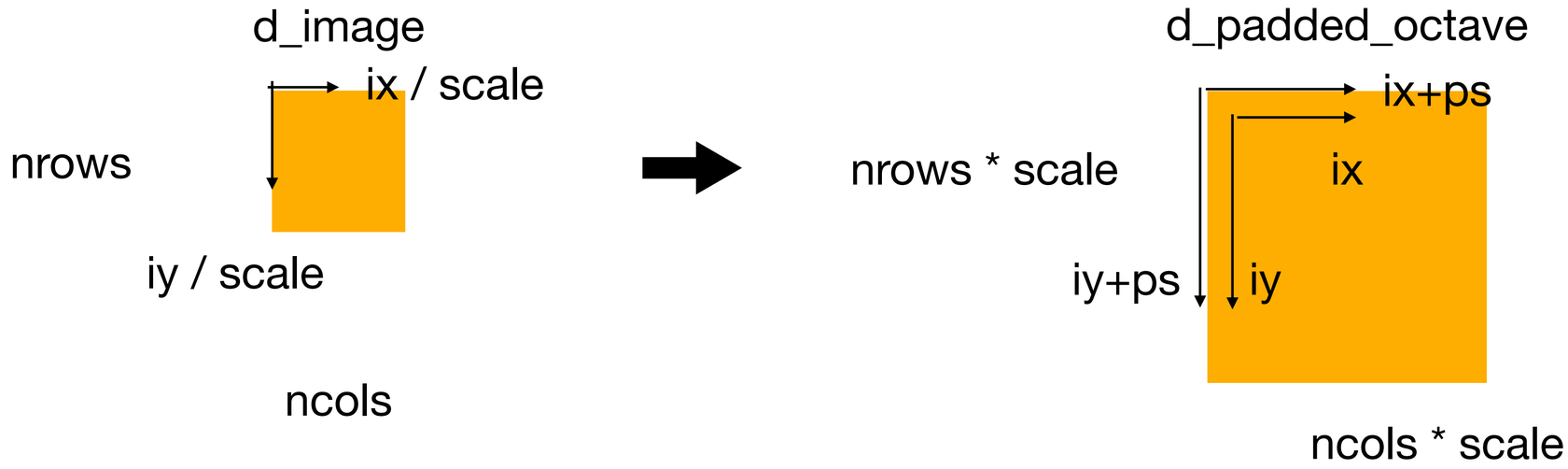
Project Overview

(2) Compute 4 Padded Octaves - host function




```
void computeAndPadOctave(float *padded_octave, float *image, const unsigned int nrows, const unsigned int ncols, const unsigned int ps, const float scale) {
    // cudaMalloc gpu image float array and gpu padded octave float array
    const unsigned  int nrows_padded_octave = nrows * scale + 2 * ps, ncols_padded_octave = ncols * scale + 2 * ps;
    //printf("nrows = %d, ncols = %d, pad_size = %d, scale = %f, nrows_padded_octave = %d, ncols_padded_octave = %d\n", nrows, ncols, ps, scale, nrows_padded
    float *d_image, *d_padded_octave;
    cudaMalloc((void**)&d_image, nrows * ncols * sizeof(float));
    cudaMalloc((void**)&d_padded_octave, nrows_padded_octave * ncols_padded_octave * sizeof(float));

    // memcpy cpu image float array to gpu image float array
    cudaMemcpy(d_image, image, nrows * ncols * sizeof(float), cudaMemcpyHostToDevice);

    // compute gpu padded octave float array
    const unsigned int nx = ncols * scale, ny = nrows * scale, dimx = 32, dimy = 32;
    dim3 block(dimx, dimy);
    dim3 grid((nx + block.x - 1) / block.x, (ny + block.y - 1) / block.y);

    gpuComputePaddedOctave << <grid, block >> > (d_padded_octave, d_image, nrows, ncols, ps, scale);

    // cudaMemcpy gpu padded octave float array to cpu padded octave float array
    cudaMemcpy(padded_octave, d_padded_octave, nrows_padded_octave * ncols_padded_octave * sizeof(float), cudaMemcpyDeviceToHost);

    // free and cudaFree
    cudaFree(d_padded_octave); cudaFree(d_image);
}
```

Project Overview (2) Compute 4 Padded Octaves - device function




```
__global__ void gpuComputePaddedOctave(float *d_padded_octave, float *d_image, const int nrows, const int ncols, const int ps, const float scale) {
    const unsigned int ix = blockIdx.x * blockDim.x + threadIdx.x;
    const unsigned int iy = blockIdx.y * blockDim.y + threadIdx.y;
    const unsigned int nrows_scale = nrows * scale, ncols_scale = ncols * scale, i = iy * ncols_scale + ix;
    if ((ix >= ncols_scale) || (iy >= nrows_scale)) return;

    // index on gpu padded octave array
    const unsigned int iy_pad = iy + ps, ix_pad = ix + ps, nrows_padded_octave = nrows_scale + 2 * ps, ncols_padded_octave = ncols_scale + 2 * ps;
    const unsigned int i_po = iy_pad * ncols_padded_octave + ix_pad; // read by row

    // index on gpu image array
    const unsigned int iy_img = iy / scale, ix_img = ix / scale;
    const unsigned int i_img = iy_img * ncols + ix_img; // write by row

    // copy value
    d_padded_octave[i_po] = d_image[i_img];
    /* ... */
}
```

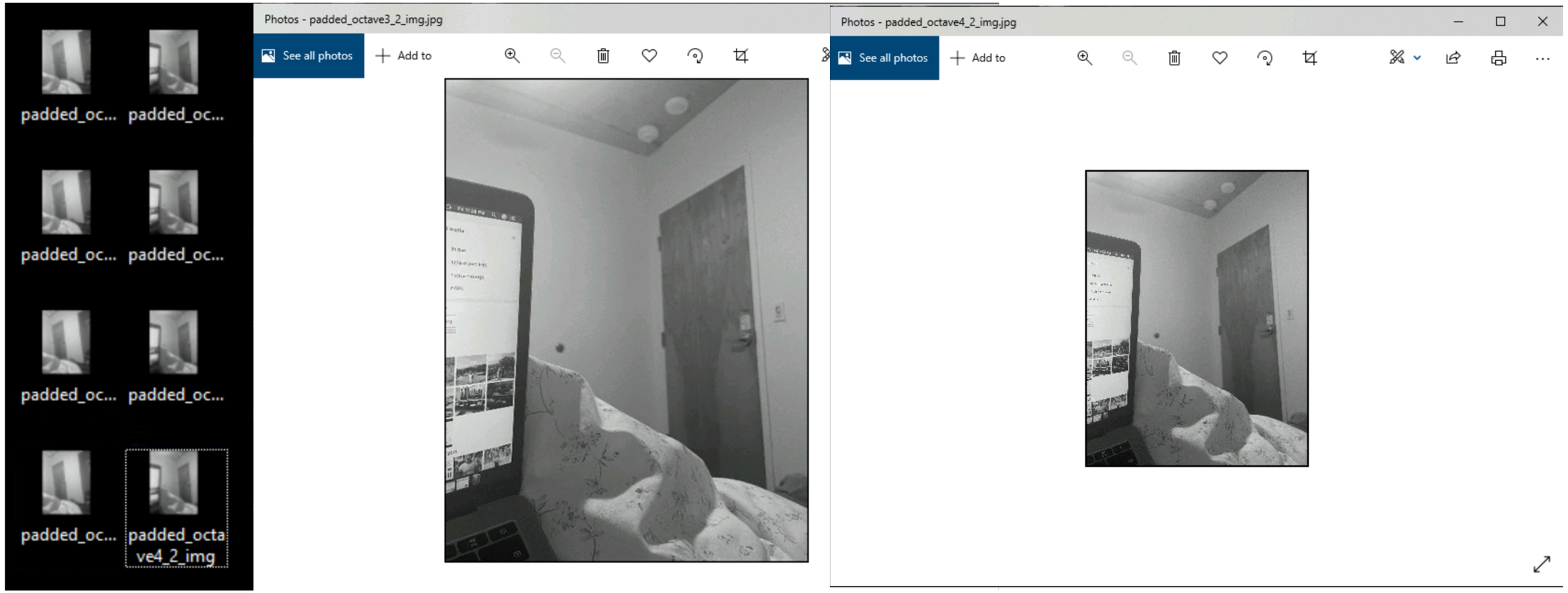
Project Overview

(2) Compute 4 Padded Octaves - save

```
void savePaddedOctaves(float *padded_octave1, float *padded_octave2, float *padded_octave3, float *padded_octave4,
    printf("saving padded octaves of image %d\n", img_id);
    const unsigned int nrows_padded_octave1 = nrows * 2 + 2 * ps, ncols_padded_octave1 = ncols * 2 + 2 * ps;
    const unsigned int nrows_padded_octave2 = nrows + 2 * ps, ncols_padded_octave2 = ncols + 2 * ps;
    const unsigned int nrows_padded_octave3 = nrows / 2 + 2 * ps, ncols_padded_octave3 = ncols / 2 + 2 * ps;
    const unsigned int nrows_padded_octave4 = nrows / 4 + 2 * ps, ncols_padded_octave4 = ncols / 4 + 2 * ps;
    Mat padded_octave1_img(nrows_padded_octave1, ncols_padded_octave1, CV_32FC1, padded_octave1);
    Mat padded_octave2_img(nrows_padded_octave2, ncols_padded_octave2, CV_32FC1, padded_octave2);
    Mat padded_octave3_img(nrows_padded_octave3, ncols_padded_octave3, CV_32FC1, padded_octave3);
    Mat padded_octave4_img(nrows_padded_octave4, ncols_padded_octave4, CV_32FC1, padded_octave4);

    char path1[100] = "", path2[100] = "", path3[100] = "", path4[100] = "";
    sprintf(path1, "C:/Users/c2lo/Desktop/padded_octave1_%d_img.jpg", img_id);
    sprintf(path2, "C:/Users/c2lo/Desktop/padded_octave2_%d_img.jpg", img_id);
    sprintf(path3, "C:/Users/c2lo/Desktop/padded_octave3_%d_img.jpg", img_id);
    sprintf(path4, "C:/Users/c2lo/Desktop/padded_octave4_%d_img.jpg", img_id);
    imwrite(path1, padded_octave1_img); imwrite(path2, padded_octave2_img);
    imwrite(path3, padded_octave3_img); imwrite(path4, padded_octave4_img);
}
```

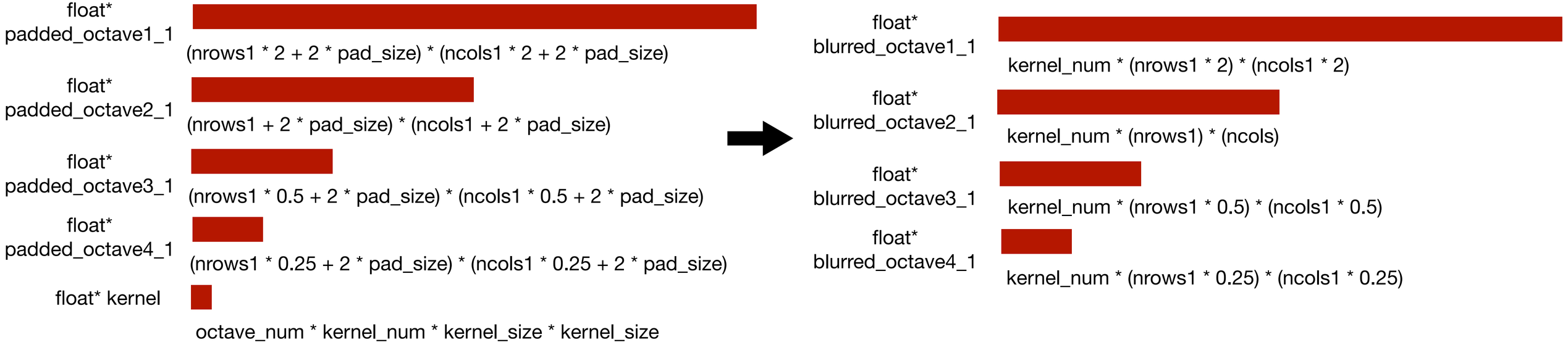
Project Overview

(2) Compute 4 Padded Octaves - result



Project Overview

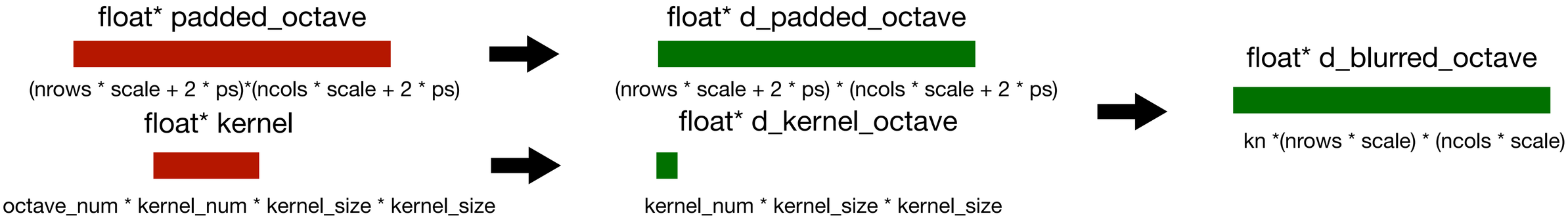
(3) Compute 20 Gaussian Blurs (4 octaves, 5 blur levels) - main




```
//3. gaussian blur
const unsigned int nrows_octave1_1 = nrows1 * 2, ncols_octave1_1 = ncols1 * 2;
const unsigned int nrows_octave2_1 = nrows1, ncols_octave2_1 = ncols1;
const unsigned int nrows_octave3_1 = nrows1 / 2, ncols_octave3_1 = ncols1 / 2;
const unsigned int nrows_octave4_1 = nrows1 / 4, ncols_octave4_1 = ncols1 / 4;
float *blurred_octave1_1 = (float*)malloc(kernel_num * nrows_octave1_1 * ncols_octave1_1 * sizeof(float));
float *blurred_octave2_1 = (float*)malloc(kernel_num * nrows_octave2_1 * ncols_octave2_1 * sizeof(float));
float *blurred_octave3_1 = (float*)malloc(kernel_num * nrows_octave3_1 * ncols_octave3_1 * sizeof(float));
float *blurred_octave4_1 = (float*)malloc(kernel_num * nrows_octave4_1 * ncols_octave4_1 * sizeof(float));
blurOctave(blurred_octave1_1, padded_octave1_1, kernel, nrows1, ncols1, kernel_num, pad_size, 2.0);
blurOctave(blurred_octave2_1, padded_octave2_1, kernel, nrows1, ncols1, kernel_num, pad_size, 1.0);
blurOctave(blurred_octave3_1, padded_octave3_1, kernel, nrows1, ncols1, kernel_num, pad_size, 0.5);
blurOctave(blurred_octave4_1, padded_octave4_1, kernel, nrows1, ncols1, kernel_num, pad_size, 0.25);
saveblurredOctaves(blurred_octave1_1, blurred_octave2_1, blurred_octave3_1, blurred_octave4_1, nrows1, ncols1, kernel_num, 1);
```

Project Overview

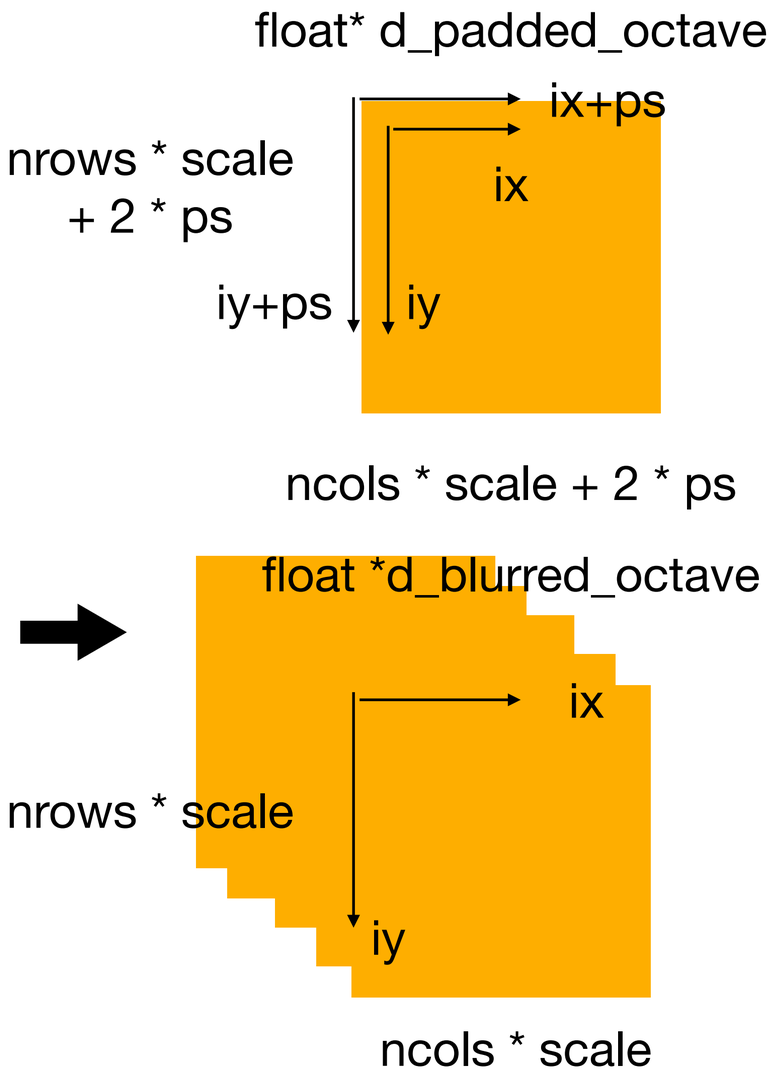
(3) Compute 20 Gaussian Blurs (4 octaves, 5 blur levels) - host function

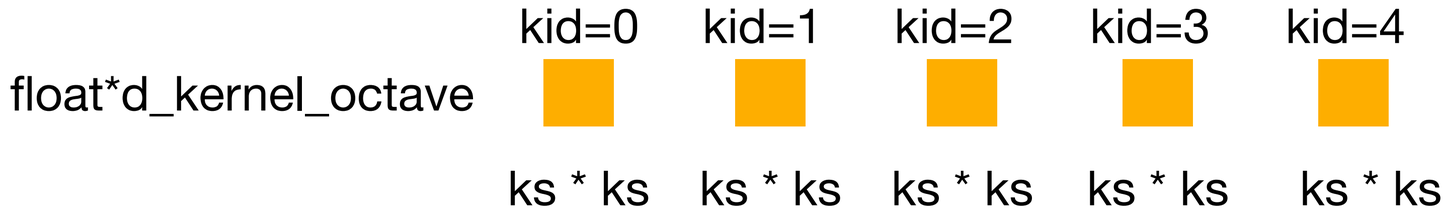



```
void blurOctave(float *blurred_octave, float *padded_octave, float* kernel, const unsigned int nrows, const unsigned int ncols, const unsigne
    // cudaMalloc gpu blurred octave float array, gpu padded octave float array, gpu kernerl float array
    float *d_blurred_octave, *d_padded_octave, *d_kernel_octave;
    const unsigned int ks = 2 * ps + 1;
    const unsigned int nrows_blurred_octave = nrows * scale, ncols_blurred_octave = ncols * scale;
    const unsigned int nrows_padded_octave = nrows * scale + 2 * ps, ncols_padded_octave = ncols * scale + 2 * ps;
    cudaMalloc((void**)&d_blurred_octave, kn * nrows_blurred_octave * ncols_blurred_octave * sizeof(float));
    cudaMalloc((void**)&d_padded_octave, nrows_padded_octave * ncols_padded_octave * sizeof(float));
    cudaMalloc((void**)&d_kernel_octave, kn * ks * ks * sizeof(float));

    // memcpy cpu padded octave and cpu kernel to gpu padded octave and gpu kernel_octave
    const unsigned int octave_id = -log2f(scale) + 1;
    cudaMemcpy(d_padded_octave, padded_octave, nrows_padded_octave * ncols_padded_octave * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_kernel_octave, &kernel[octave_id * kn * ks * ks], kn * ks * ks * sizeof(float), cudaMemcpyHostToDevice);
    /* ... */

    // compute gaussian blur on gpu
    const unsigned int nx = ncols * scale, ny = nrows * scale, dimx = 32, dimy = 32;
    dim3 block(dimx, dimy);
    dim3 grid((nx + block.x - 1) / block.x, (ny + block.y - 1) / block.y);
    gpuBlurOctave << <grid, block >> > (d_blurred_octave, d_padded_octave, d_kernel_octave, nrows, ncols, kn, ps, scale);

    // cudaMemcpy gpu padded octave float array to cpu padded octave float array
    cudaMemcpy(blurred_octave, d_blurred_octave, kn * nrows_blurred_octave * ncols_blurred_octave * sizeof(float), cudaMemcpyDeviceToHost);

    // free and cudaFree
    cudaFree(d_blurred_octave); cudaFree(d_padded_octave); cudaFree(d_kernel_octave);
}
```

Project Overview

(3) Compute 20 Gaussian Blurs (4 octaves, 5 blur levels) - device function






```
__global__ void gpuBlurOctave(float *d_blurred_octave, float *d_padded_octave, float *d_kernel_octave, const unsigned
    // index on gpu padded octave array
    const unsigned int ix = blockIdx.x * blockDim.x + threadIdx.x;
    const unsigned int iy = blockIdx.y * blockDim.y + threadIdx.y;
    const unsigned int nrows_scale = nrows * scale, ncols_scale = ncols * scale;
    if ((ix >= ncols_scale) || (iy >= nrows_scale)) return;
    const unsigned int iy_pad = iy + ps, ix_pad = ix + ps;
    const unsigned int ks = 2 * ps + 1;
    const unsigned int nrows_padded_octave = nrows_scale + 2 * ps, ncols_padded_octave = ncols_scale + 2 * ps;
    // ...

    // copy value
    for (unsigned int kid = 0; kid < kn; kid++) {
        const unsigned int i = kid * nrows_scale * ncols_scale + iy * ncols_scale + ix;
        float tmp = 0;
        for (int j = -ps; j <= ps; j++) {
            for (int k = -ps; k <= ps; k++) {
                const unsigned int i_ko = kid * ks * ks + (k + ps) * ks + (j + ps);
                const unsigned int i_po = (iy_pad + k) * ncols_padded_octave + (ix_pad + j);
                tmp += d_kernel_octave[i_ko] * d_padded_octave[i_po];
                /* ... */
            }
        }
        d_blurred_octave[i] = tmp;
    }
}
```

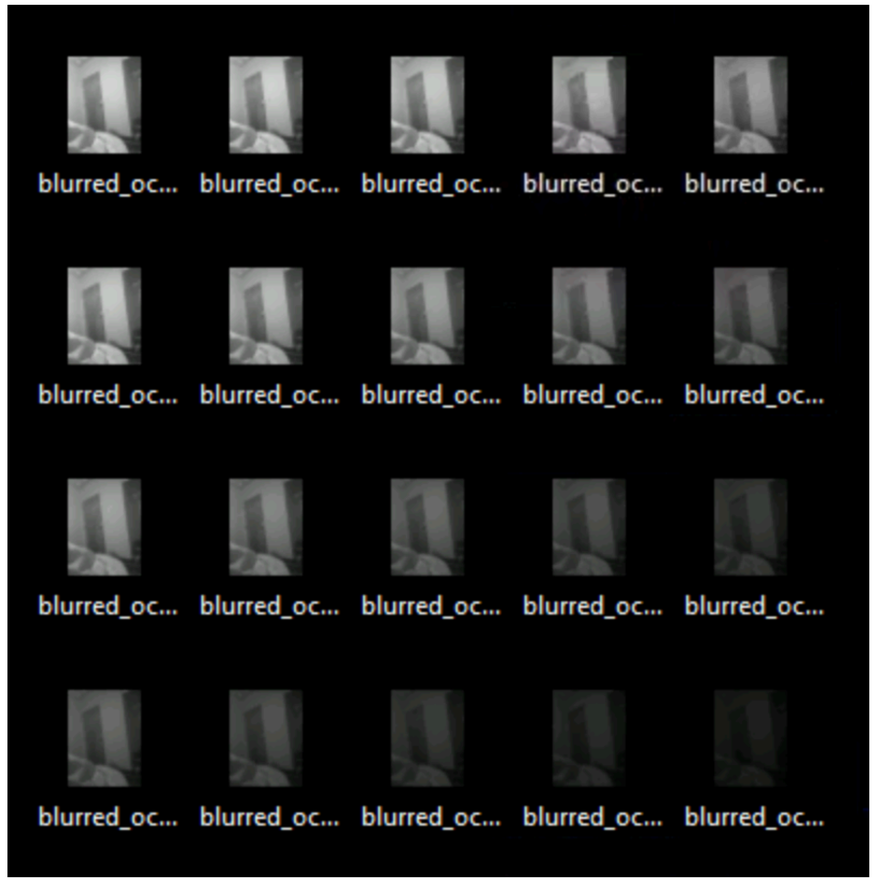
Project Overview

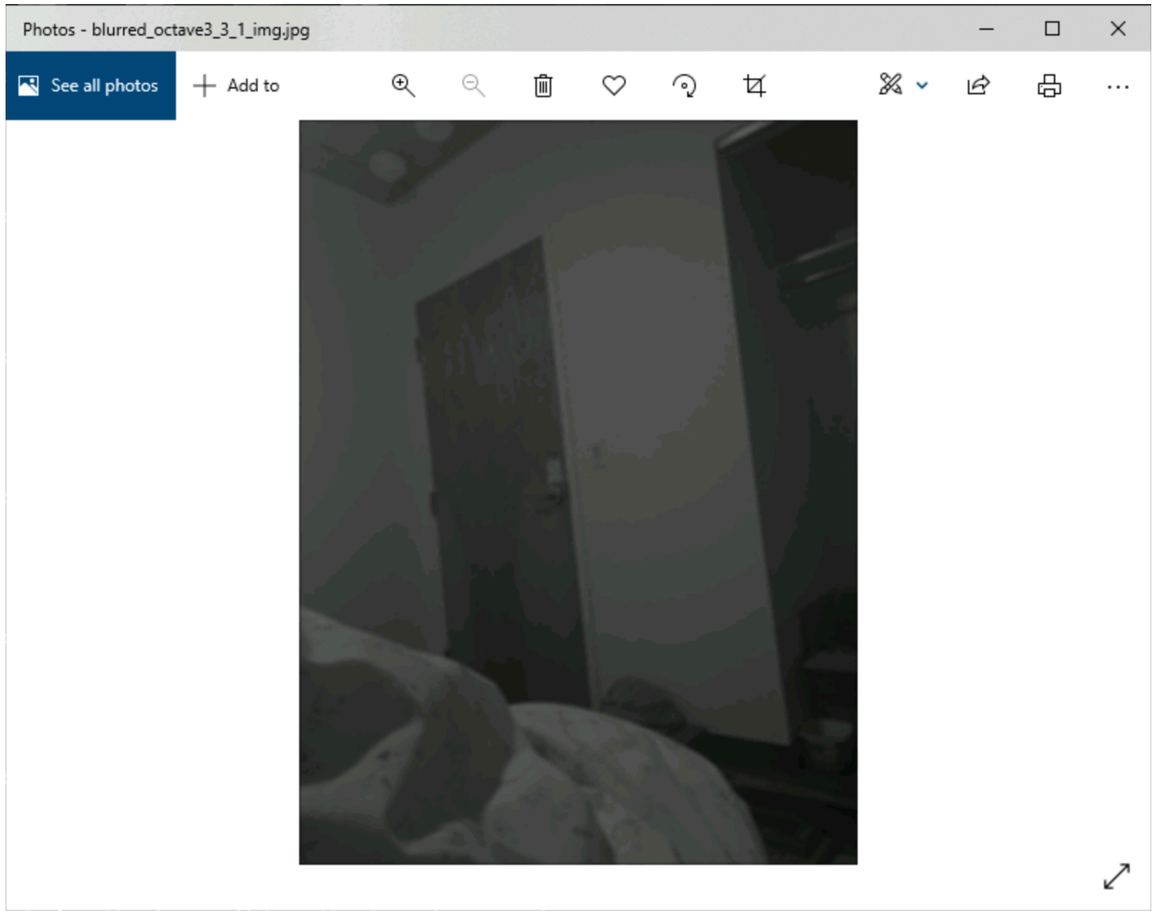
(3) Compute 20 Gaussian Blurs (4 octaves, 5 blur levels) - save

```
void saveblurredOctaves(float *blurred_octave1, float *blurred_octave2, float *blurred_octave3, float *blurred_octave4, const unsigned int nrows, const unsigne
    printf("saving blurredd octaves of image %d\n", img_id);
    for (int kid = 0; kid < kn; kid++) {
        const unsigned int nrows_blurred_octave1 = nrows * 2, ncols_blurred_octave1 = ncols * 2;
        const unsigned int nrows_blurred_octave2 = nrows, ncols_blurred_octave2 = ncols;
        const unsigned int nrows_blurred_octave3 = nrows / 2, ncols_blurred_octave3 = ncols / 2;
        const unsigned int nrows_blurred_octave4 = nrows / 4, ncols_blurred_octave4 = ncols / 4;

        float *blurred_octave1_k = (float*)malloc(nrows_blurred_octave1 * ncols_blurred_octave1 * sizeof(float));
        float *blurred_octave2_k = (float*)malloc(nrows_blurred_octave2 * ncols_blurred_octave2 * sizeof(float));
        float *blurred_octave3_k = (float*)malloc(nrows_blurred_octave3 * ncols_blurred_octave3 * sizeof(float));
        float *blurred_octave4_k = (float*)malloc(nrows_blurred_octave4 * ncols_blurred_octave4 * sizeof(float));
        memcpy(blurred_octave1_k, &blurred_octave1[kid * nrows_blurred_octave1 * ncols_blurred_octave1], nrows_blurred_octave1 * ncols_blurred_octave1 * sizeof
        memcpy(blurred_octave2_k, &blurred_octave2[kid * nrows_blurred_octave2 * ncols_blurred_octave2], nrows_blurred_octave2 * ncols_blurred_octave2 * sizeof
        memcpy(blurred_octave3_k, &blurred_octave3[kid * nrows_blurred_octave3 * ncols_blurred_octave3], nrows_blurred_octave3 * ncols_blurred_octave3 * sizeof
        memcpy(blurred_octave4_k, &blurred_octave4[kid * nrows_blurred_octave4 * ncols_blurred_octave4], nrows_blurred_octave4 * ncols_blurred_octave4 * sizeof
        Mat blurred_octave1_img(nrows_blurred_octave1, ncols_blurred_octave1, CV_32FC1, blurred_octave1_k);
        Mat blurred_octave2_img(nrows_blurred_octave2, ncols_blurred_octave2, CV_32FC1, blurred_octave2_k);
        Mat blurred_octave3_img(nrows_blurred_octave3, ncols_blurred_octave3, CV_32FC1, blurred_octave3_k);
        Mat blurred_octave4_img(nrows_blurred_octave4, ncols_blurred_octave4, CV_32FC1, blurred_octave4_k);

        char path1[100] = "", path2[100] = "", path3[100] = "", path4[100] = "";
        sprintf(path1, "C:/Users/c2lo/Desktop/blurred_octave1_%d_%d_img.jpg", kid, img_id);
        sprintf(path2, "C:/Users/c2lo/Desktop/blurred_octave2_%d_%d_img.jpg", kid, img_id);
        sprintf(path3, "C:/Users/c2lo/Desktop/blurred_octave3_%d_%d_img.jpg", kid, img_id);
        sprintf(path4, "C:/Users/c2lo/Desktop/blurred_octave4_%d_%d_img.jpg", kid, img_id);
        imwrite(path1, blurred_octave1_img); imwrite(path2, blurred_octave2_img);
        imwrite(path3, blurred_octave3_img); imwrite(path4, blurred_octave4_img);
    }
}
```

Project Overview

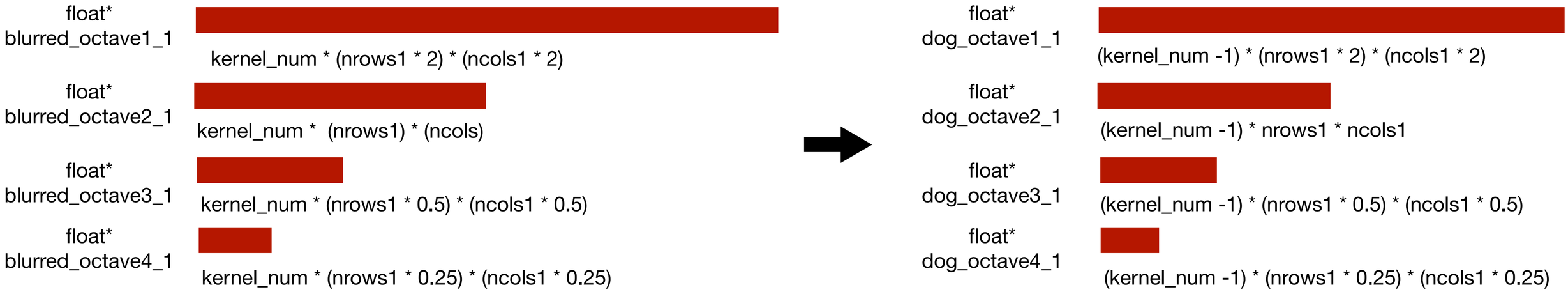
(3) Compute 20 Gaussian Blurs (4 octaves, 5 blur levels) - result





Project Overview

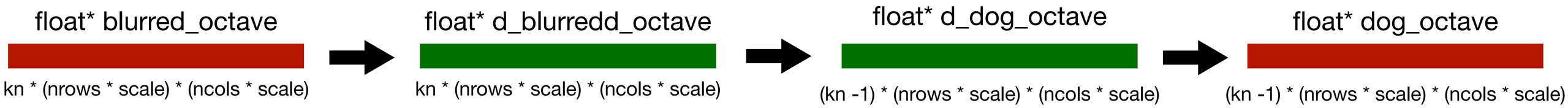
(4) Compute 16 DOGs (4 octaves, 4 DOGs) - main




```
//4. laplacian of gaussian
float *dog_octave1_1 = (float*)malloc((kernel_num - 1) * nrows_octave1_1 * ncols_octave1_1 * sizeof(float));
float *dog_octave2_1 = (float*)malloc((kernel_num - 1) * nrows_octave2_1 * ncols_octave2_1 * sizeof(float));
float *dog_octave3_1 = (float*)malloc((kernel_num - 1)* nrows_octave3_1 * ncols_octave3_1 * sizeof(float));
float *dog_octave4_1 = (float*)malloc((kernel_num - 1) * nrows_octave4_1 * ncols_octave4_1 * sizeof(float));
computeDogOctave(dog_octave1_1, blurred_octave1_1, nrows1, ncols1, kernel_num, 2.0);
computeDogOctave(dog_octave2_1, blurred_octave2_1, nrows1, ncols1, kernel_num, 1.0);
computeDogOctave(dog_octave3_1, blurred_octave3_1, nrows1, ncols1, kernel_num, 0.5);
computeDogOctave(dog_octave4_1, blurred_octave4_1, nrows1, ncols1, kernel_num, 0.25);
saveDogOctaves(dog_octave1_1, dog_octave2_1, dog_octave3_1, dog_octave4_1, nrows1, ncols1, kernel_num, 1);
```

Project Overview

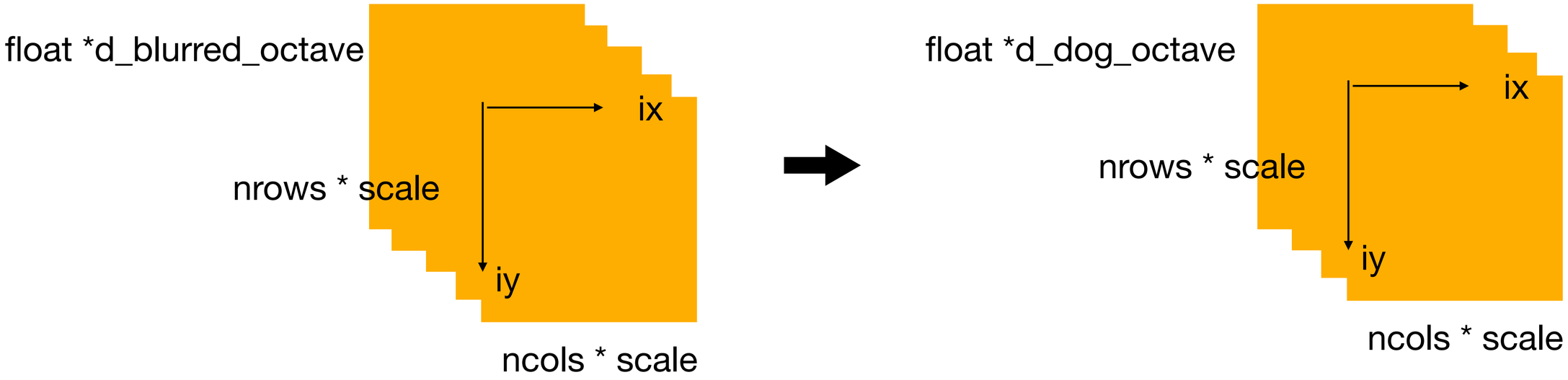
(4) Compute 16 DOGs (4 octaves, 4 DOGs) - host function




```
void computeDogOctave(float *dog_octave, float *blurred_octave, const unsigned int nrows, const unsigned int ncols, const unsigned int kn, cc
    // cudaMalloc gpu blurred octave and gpu dog octave
    float *d_dog_octave, *d_blurred_octave;
    const unsigned int nrows_blurred_octave = nrows * scale, ncols_blurred_octave = ncols * scale;
    cudaMalloc((void**) &d_dog_octave, (kn - 1) * nrows_blurred_octave * ncols_blurred_octave * sizeof(float));
    cudaMalloc((void**) &d_blurred_octave, kn * nrows_blurred_octave * ncols_blurred_octave * sizeof(float));

    // memcpy cpu blurredd octave to device
    cudaMemcpy(d_blurred_octave, blurred_octave, kn * nrows_blurred_octave * ncols_blurred_octave * sizeof(float), cudaMemcpyHostToDevice);

    // compute difference of gaussian
    const unsigned int nx = ncols * scale, ny = nrows * scale, dimx = 32, dimy = 32;
    dim3 block(dimx, dimy);
    dim3 grid((nx + block.x - 1) / block.x, (ny + block.y - 1) / block.y);
    gpuComputeDogOctave << <grid, block >> > (d_dog_octave, d_blurred_octave, nrows, ncols, kn, scale);
    cudaDeviceSynchronize();

    // cudaMemcpy gpu dog octave host
    cudaMemcpy(dog_octave, d_dog_octave, (kn-1) * nrows_blurred_octave * ncols_blurred_octave * sizeof(float), cudaMemcpyDeviceToHost);

    // cudaFree
    cudaFree(d_dog_octave); cudaFree(d_blurred_octave);
}
```

Project Overview

(4) Compute 16 DOGs (4 octaves, 4 DOGs) - device function




```
__global__ void gpuComputeDogOctave(float *d_dog_octave, float *d_blurred_octave, const unsigned int r
    // index on gpu dog octave array
    const unsigned int ix = blockIdx.x * blockDim.x + threadIdx.x;
    const unsigned int iy = blockIdx.y * blockDim.y + threadIdx.y;
    const unsigned int nrows_scale = nrows * scale, ncols_scale = ncols * scale;
    if ((ix >= ncols_scale) || (iy >= nrows_scale)) return;

    // substract and assign
    for (unsigned int kid = 0; kid < kn - 1 ; kid++) {
        const unsigned int i = kid * nrows_scale * ncols_scale + iy * ncols_scale + ix;
        d_dog_octave[i] = d_blurred_octave[i] - d_blurred_octave[i + nrows_scale * ncols_scale];
    }
}
```

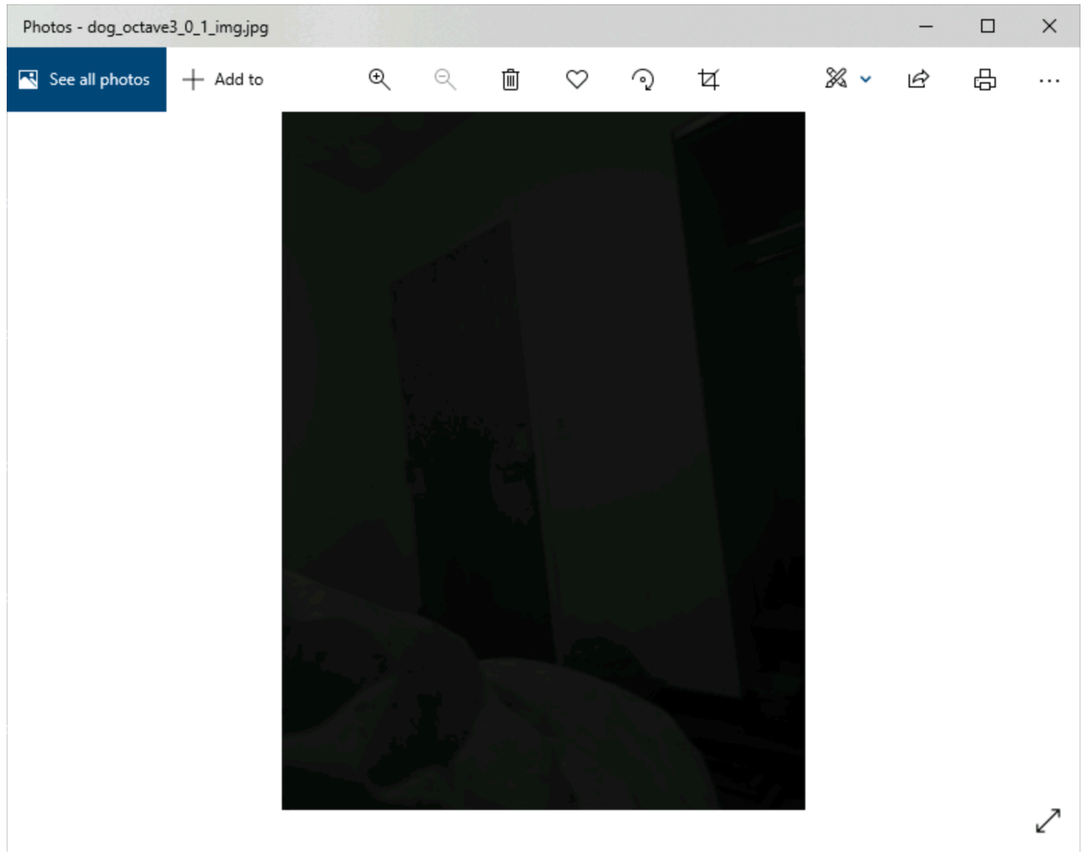
Project Overview

(4) Compute 16 DOGs (4 octaves, 4 DOGs) - save

```
void saveDogOctaves(float *dog_octave1, float *dog_octave2, float *dog_octave3, float *dog_octave4, const unsigned int nrows, const unsigned int n
    printf("saving dog octaves of image %d\n", img_id);
    for (unsigned int kid = 0; kid < kn-1; kid++) {
        const unsigned int nrows_dog_octave1 = nrows * 2, ncols_dog_octave1 = ncols * 2;
        const unsigned int nrows_dog_octave2 = nrows, ncols_dog_octave2 = ncols;
        const unsigned int nrows_dog_octave3 = nrows / 2, ncols_dog_octave3 = ncols / 2;
        const unsigned int nrows_dog_octave4 = nrows / 4, ncols_dog_octave4 = ncols / 4;

        float *dog_octave1_k = (float*)malloc(nrows_dog_octave1 * ncols_dog_octave1 * sizeof(float));
        float *dog_octave2_k = (float*)malloc(nrows_dog_octave2 * ncols_dog_octave2 * sizeof(float));
        float *dog_octave3_k = (float*)malloc(nrows_dog_octave3 * ncols_dog_octave3 * sizeof(float));
        float *dog_octave4_k = (float*)malloc(nrows_dog_octave4 * ncols_dog_octave4 * sizeof(float));
        memcpy(dog_octave1_k, &dog_octave1[kid * nrows_dog_octave1 * ncols_dog_octave1], nrows_dog_octave1 * ncols_dog_octave1 * sizeof(float));
        memcpy(dog_octave2_k, &dog_octave2[kid * nrows_dog_octave2 * ncols_dog_octave2], nrows_dog_octave2 * ncols_dog_octave2 * sizeof(float));
        memcpy(dog_octave3_k, &dog_octave3[kid * nrows_dog_octave3 * ncols_dog_octave3], nrows_dog_octave3 * ncols_dog_octave3 * sizeof(float));
        memcpy(dog_octave4_k, &dog_octave4[kid * nrows_dog_octave4 * ncols_dog_octave4], nrows_dog_octave4 * ncols_dog_octave4 * sizeof(float));
        Mat dog_octave1_img(nrows_dog_octave1, ncols_dog_octave1, CV_32FC1, dog_octave1_k);
        Mat dog_octave2_img(nrows_dog_octave2, ncols_dog_octave2, CV_32FC1, dog_octave2_k);
        Mat dog_octave3_img(nrows_dog_octave3, ncols_dog_octave3, CV_32FC1, dog_octave3_k);
        Mat dog_octave4_img(nrows_dog_octave4, ncols_dog_octave4, CV_32FC1, dog_octave4_k);

        char path1[100] = "", path2[100] = "", path3[100] = "", path4[100] = "";
        sprintf(path1, "C:/Users/c2lo/Desktop/dog_octave1_%d_%d_img.jpg", kid, img_id);
        sprintf(path2, "C:/Users/c2lo/Desktop/dog_octave2_%d_%d_img.jpg", kid, img_id);
        sprintf(path3, "C:/Users/c2lo/Desktop/dog_octave3_%d_%d_img.jpg", kid, img_id);
        sprintf(path4, "C:/Users/c2lo/Desktop/dog_octave4_%d_%d_img.jpg", kid, img_id);
        imwrite(path1, dog_octave1_img); imwrite(path2, dog_octave2_img);
        imwrite(path3, dog_octave3_img); imwrite(path4, dog_octave4_img);
    }
}
```

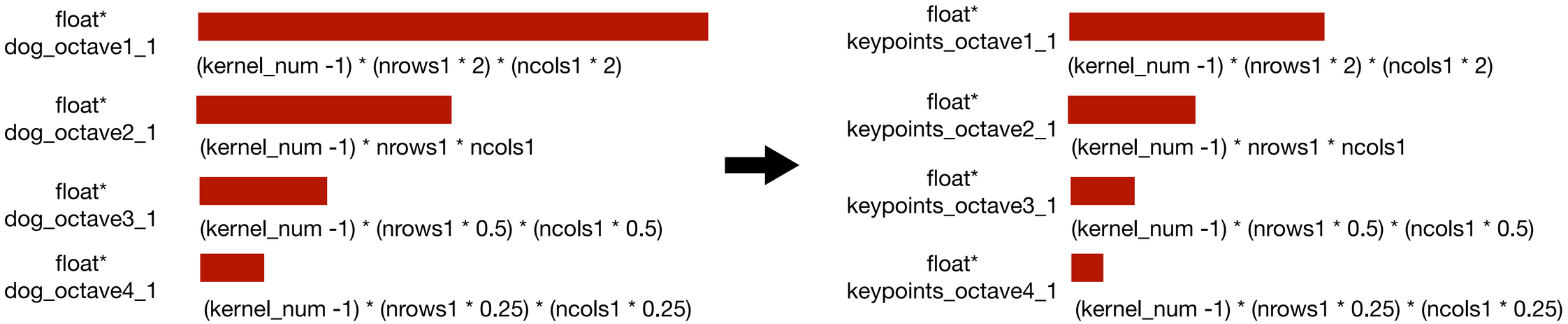
Project Overview

(4) Compute 16 DOGs (4 octaves, 4 DOGs) - main



Project Overview

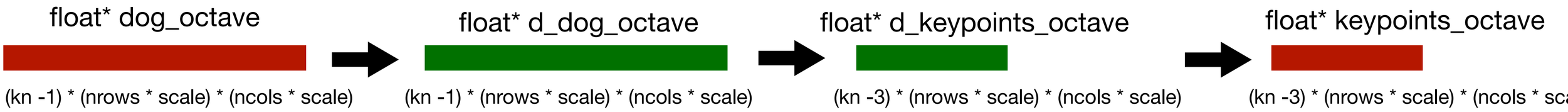
(5) Compute Key Points - main




```
//5. find key points
unsigned short *keypoints_octave1_1 = (unsigned short*)malloc((kernel_num - 3) * nrows_octave1_1 * ncols_octave1_1 * sizeof(unsigned short));
unsigned short *keypoints_octave2_1 = (unsigned short*)malloc((kernel_num - 3) * nrows_octave2_1 * ncols_octave2_1 * sizeof(unsigned short));
unsigned short *keypoints_octave3_1 = (unsigned short*)malloc((kernel_num - 3)* nrows_octave3_1 * ncols_octave3_1 * sizeof(unsigned short));
unsigned short *keypoints_octave4_1 = (unsigned short*)malloc((kernel_num - 3) * nrows_octave4_1 * ncols_octave4_1 * sizeof(unsigned short));
findKeypointsOctave(keypoints_octave1_1, dog_octave1_1, nrows1, ncols1, kernel_num, 2.0);
findKeypointsOctave(keypoints_octave2_1, dog_octave2_1, nrows1, ncols1, kernel_num, 1.0);
findKeypointsOctave(keypoints_octave3_1, dog_octave3_1, nrows1, ncols1, kernel_num, 0.5);
findKeypointsOctave(keypoints_octave4_1, dog_octave4_1, nrows1, ncols1, kernel_num, 0.25);
```

Project Overview

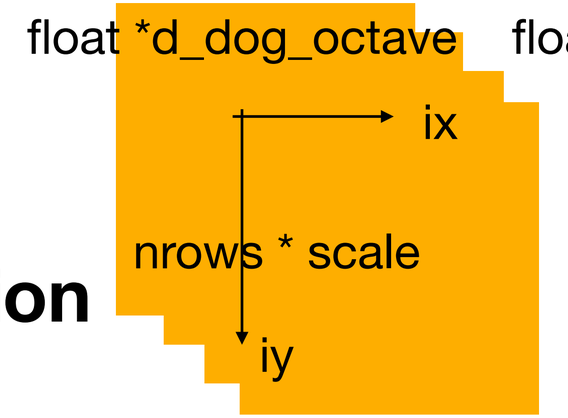
(5) Compute Key Points - host function

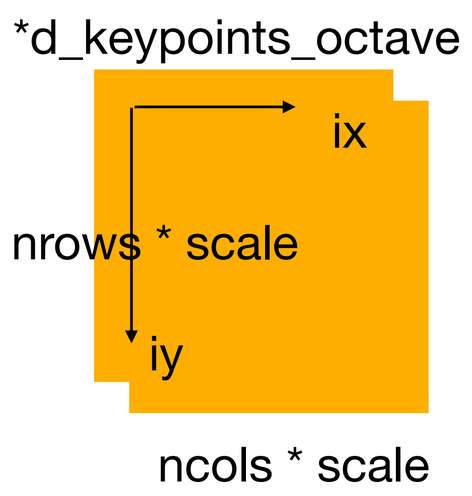



```
void findKeypointsOctave(unsigned short *keypoints_octave, float *dog_octave, const unsigned int nrows, const unsigned int ncols, const unsigned int kn, co
    // cudaMalloc gpu keypoints octave and gpu dog octave
    unsigned short *d_keypoints_octave;
    float *d_dog_octave;
    const unsigned int nrows_dog_octave = nrows * scale, ncols_dog_octave = ncols * scale;
    cudaMalloc((void**) &d_keypoints_octave, (kn - 3) * nrows_dog_octave * ncols_dog_octave * sizeof(unsigned short));
    cudaMalloc((void**) &d_dog_octave, (kn - 1) * nrows_dog_octave * ncols_dog_octave * sizeof(float));

    // memcpy cpu dog octave to device
    cudaMemcpyAsync(d_dog_octave, dog_octave, (kn - 1) * nrows_dog_octave * ncols_dog_octave * sizeof(float), cudaMemcpyHostToDevice);

    // find keypoints
    const unsigned int nx = ncols * scale, ny = nrows * scale, dimx = 32, dimy = 32;
    dim3 block(dimx, dimy);
    dim3 grid((nx + block.x - 1) / block.x, (ny + block.y - 1) / block.y);
    printf("nx = %d, ny = %d\n", nx, ny);
    gpuFindKeypointsOctave << <grid, block >> > (d_keypoints_octave, d_dog_octave, nrows, ncols, kn, scale);
    cudaDeviceSynchronize();

    // cudaMemcpy gpu keypoints octave to host
    cudaMemcpyAsync(keypoints_octave, d_keypoints_octave, (kn - 3) *  nrows_dog_octave * ncols_dog_octave * sizeof(unsigned short), cudaMemcpyDeviceToHost)

    // cudaFree
    cudaFree(d_keypoints_octave); cudaFree(d_dog_octave);
}
```

Project Overview

(5) Compute Key Points - device function






```
__global__ void gpuFindKeypointsOctave(unsigned short *d_keypoints_octave, float *d_dog_octave, const unsigned int nrows, const unsigned int ncols, const u
    // index on gpu dog octave array
    const unsigned int ix = blockIdx.x * blockDim.x + threadIdx.x;
    const unsigned int iy = blockIdx.y * blockDim.y + threadIdx.y;
    const unsigned int nrows_scale = nrows * scale, ncols_scale = ncols * scale;
    if ((ix >= ncols_scale) || (iy >= nrows_scale)) return;
    if ((ix == ncols_scale - 1) && (iy == nrows_scale - 1)) printf("****************************************\n");
    //if (threadIdx.x==0 && threadIdx.y==0) printf("ix = %d, ncols_scale = %d, blockIdx.x = %d, threadIdx.x = %d, iy = %d, nrows_scale = %d, blockIdx.y = %

    // scan neigbors and determine if it is keypoint
    for (unsigned int kid = 1; kid < kn - 2; kid++) {
        const unsigned int i = kid * nrows_scale * ncols_scale + iy * ncols_scale + ix;
        bool is_max = true, is_min = true;
        for (int l = -1; l <= 1; l++) { // previous layer to next layer
            for (int j = -1; j <= 1; j++) { //previous column to next column
                for (int k = -1; k <= 1; k++) { // previous row to next row
                    // index of neighbor
                    //if (threadIdx.x == 0 && threadIdx.y == 0) printf("ix = %d, iy = %d, kid = %d, l = %d, j = %d, k = %d\n", ix, iy, kid, l, j, k);
                    const unsigned int i_n = (kid + l) * nrows_scale * ncols_scale + (iy + k) * ncols_scale + (ix + j);
                    if (d_dog_octave[i_n]>d_dog_octave[i]) is_max = false;
                    if (d_dog_octave[i_n]<d_dog_octave[i]) is_min = false;
                }
            }
        }

        if (is_max == true || is_min == true) d_keypoints_octave[i] = 255;
        else d_keypoints_octave[i] = 0;
        if (threadIdx.x == 0 && threadIdx.y == 0) printf("kid = %d, ix = %d, iy = %d, d_keypoints_octave[i] = %d\n", kid, ix, iy, d_keypoints_octave[i]);
    }
}
```

References

- Paper of SIFT ……..
- SIFT written in Python https://github.com/taem98/SIFT/
- Parallel_SIFT written in C++ https://github.com/Alvinosaur/Parallel_SIFT
- Introduction to SIFT

  https://aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/
- OpenCV set up videos