[These lecture notes complement the slides. You should read both!]

Suppose that we have computed a scale space using a $\nabla^2 g_\sigma(x, y)$ filter. I argued last lecture that such a scale space will give local maxima and local minima over (x, y, σ) when there is a box-like intensity pattern present (sometimes called a "blob"). The box either can be brighter than its background (this gives a local minimum) or darker than its local background (this gives a local maximum). The maxima and minima occur at specific scales which are related to the size of the box.

As we saw with the examples in the slides from last lecture, local maxima and minima arise from other image intensity patterns – not just 2D boxs (squares). It is common to refer to such local maxima and minima of the filtered image as *keypoints*. These are similar to Harris corner points, but now we are specifically defining them to be local maxes and mins of the filtered image in a 3D scale space. Keypoints have a position and a scale.

Note: With Harris corners, we examined maxima of the Harris measure in 2D for a particular scale σ. For Harris corners, we did *not* argue that the maxima over scale should be characteristic of the "size" of the image pattern in any sense. Neither a corner nor a step edge has a natural scale. By contrast shapes like bars, boxes, disks, etc do have a naturarl scale, namely the width (or half-width).

SIFT: Scale invariant feature transform

For each keypoint, we have a position (x, y) and a scale σ. It is common to define a *descriptor* of the image in the neighborhood of each keypoint. Each keypoint has a scale σ so we use the intensity structure in the scale space at that scale σ and in a local neighborhood of the keypoint position, where the definition of "local" depends on σ. How can we describe the local intensity structure ? Let's sketch out one method, called SIFT, which was invented about 10 years ago by Dave Lowe and his students at UBC and which has been very popular.

Assume we have found a keypoint at (x_0, y_0, σ_0) which is a local maximum (or minimum) in the $\nabla^2 g_\sigma(x, y) * I(x, y)$. We ensure that this local peak is sufficiently different from its neighbors and sufficiently different from zero, i.e. it is it is well localized and it is a large peak. We now want somehow to describe the intensities in the (x, y) neighborhood of the keypoint at the particular scale σ in scale space.

Find a dominant direction

Rather than working with the image intensities themselves, SIFT constructs a local descriptor of the gradient vectors $\nabla g_\sigma(x, y, \sigma_0) * I(x, y)$ in the (x, y) neighborhood of the keypoint. Define a square of size say $6\sigma_0 \times 6\sigma_0$. Take the gradient vectors in this neighborhood and bin them by direction, by computing an *orientation histogram* which adds up the magnitudes of the gradients at each (quantized) orientation:

```
//  Let the number of angular bins be numAngleBins in degrees.
//  Lowe used 24 angular bins, so binWidth was 15 degrees.

orientationHist = zeros(numAngleBins)  % initializes to zero
for each (x,y) in Ngd(x0,y0, sig)
```

```
//  gradient vector is  grad (I * g_sigma)(x,y)
  theta = direction of gradient vector
  binTheta = round(theta / binWidth)
  len      = length of gradient vector
  orientationHist[ binTheta ] += len
```

You might also want to weight the lengths by the distance $|(x - x_0, y - y_0)|$ using a Gaussian weighting. In fact, SIFT does this.

The orientation histogram will have maximum at some direction, called the *dominant orientation*. If there are multiple maxima that are similar in value then there are multiple dominant orientations. In that case, one creates multiple feature descriptors for this keypoint (see next), namely one for each dominant orientation. See the slides for examples.

Feature descriptor

The orientation histogram is useful for defining a standard orientation for the intensity gradients near the keypoint. However, it is of limited use for describing the distribution of these gradients since it ignores the (x, y) positions, relative to the dominant direction. SIFT tries to keep some of the information about the spatial arrangement of the gradients, as follows.

Once a dominant orientation for a keypoint has been found, SIFT defines a rotated square of size $6\sigma \times 6\sigma$, that is oriented parallel to the dominant direction. Note that the width of the square is the same as the width used to find the dominant direction. The gradient vectors in the square are then used to compute a 16×16 grid of gradient vectors which sample the square. These gradient vectors can be computed using $\nabla I(x, y) * g_\sigma(x, y)$ which is defined on the pixel grid, and then interpolating from nearest neighbors.

This 16×16 grid, which is aligned with the dominant orientation, is then partitioned into a 4×4 array of 4×4 subgrids. (See slides.) For each of the subgrids, an orientation histogram with 8 orientations of 45 degrees each is constructed (rather than 24 orientations of 15 degrees, which is what was used to find the dominant orientation for the whole neigbhorhood). These $4 \times 4 = 16$ orientation histograms, with 8 bins each, define a 128 dimensional "feature descriptor". Note that we have reduced the number of dimensions from 512 ($16 \times 16 \times 2$) to 128.

An important property of these feature descriptors is that they do not change when the image is resized or rotated (i.e. they are rotationally invariant and scale invariant, respectively). These are important properties, since two images of the same scene might be somewhat rotated or scaled relative to each other. For example, the images might be shot with different cameras (or with different parameter settings of the same camera). While rotational invariance was used by several techniques prior to SIFT. SIFT is (claimed to be) the first feature descriptor that is designed to be scale invariant.

Using SIFT features for image indexing (or recognition)

Suppose we have a database of images, sometimes referred to as "training" images. We are given a new image and we would like to find whether the new image is similar to one of the images in the database. How can we do this?

For each image in the database, we compute the keypoints and the SIFT feature descriptor. For each of these SIFT features (a 128-vector), we have a vector (descriptor, imageID, $x, y, \sigma, \theta_{dominant}$).

We create a data structure that represents all these SIFT features – namely points in a 128-dimensional space – and algorithms for indexing into this data structure, namely given a new SIFT feature, find a set of similar SIFT features in the space, i.e. SIFT features that were in the training set. (This is a very general problem, and I am not going to give even a single example of how to do this.)

The SIFT features are often called *keypoints* or *keys*, and this term is helpful for understanding how they are used. Think of the a hash function, which maps keys to values. In our problem here, the keys are the SIFT feature descriptors, and the values are the vectors (imageID, $x, y, \sigma, \theta_{dominant}$).

How are the SIFT features used for indexing. Given a new image, we compute the SIFT features of that image. For each SIFT feature F_i in our new image, we find all the features in the training set that are with a threshold distance τ from the new feature (where the definition of distance is omitted here). Each of these nearby features F_j came from some image in the database (the value – see above). So, for each feature in the new image, we can index a set of candidate database images that contain a SIFT feature that is similar to F_i.

What to do next? One naive idea is to take these sets of data base images (one set for each F_i) and vote, namely cast one vote for image J if F_i is close to feature F_j and F_j belongs to (maps to) image J. Then, the database image with the most votes is the one that is chosen.

The above scheme is simple. However, it doesn't work well since it ignores all spatial relations between keypoints. One way to improve the above scheme is to use clusters of keypoints in the new image, and match these clusters. For example, take a triplet of nearby keyoints in the new image. Each of the three keypoints will generate a set of matching features descriptors in the database, with each matching feature belonging to a image. For that triplet to cast a vote for an image in the database, the three sets of matching features must have that image in common. Moreover, because each point in the triplet of features has a position (we get a triangle) and a scale in the new image, the matching features must obey a similar relationship on their scale and position. If the features scales F_1, F_2, F_3 are related by some ratio $s_1 : s_2 : s_3$ in the original image, then this contrains the scales of the matching features in the database image. Such contraints allow us to prune away many possibilities.

[ASIDE: For more details, see David Lowe's 2004 paper in the International Journal of Computer Vision.]

Coarse-to-fine image registration

Before moving on to our next topic, I will briefly return to the image registration problem, and mention how scale space can be used to solve this problem too.

Recall the image registration problem: given two images $I(x, y)$ and $J(x, y)$, we wish to find a translation (h_x, h_y) such that $I(x + h_x, y + h_y) \approx J(x, y)$ in the neighborhood of (x, y), i.e. the h vector may vary with x, y. In deriving the Lucas-Kanade method in lecture 11, we used a first order model for the intensities of $I(x, y)$ in local neighborhoods. This required that the translation distance $|(h_x, h_y)|$ be less than the radius σ_I of the neighborhood.[1] This implies that we need to choose a neighborhood scale σ_I that is much greater than h.

[1]Why? The first order model says that the intensity is linear. But if the intensity were linear over the whole neighborhood, then the intensity gradient would be constant and the second moment matrix would have a zero eigenvalue. We would not be able to solve uniquely for (h_x, h_y)!

There are problems with using a large scale, however. One problem is related to what we saw in Canny's edge analysis in lecture 9. If we used a large scale then we got relatively poor localization. For registration, the problem is that the blurring the image with a large σ reduces the image noise (good) but it also smooths out the the gradient field. And it is the details in the gradient field that are useful for precise localization.

A second problem with large σ's is that the formulation of the registration algorithm assumes that (h_x, h_y) is constant over the integration neighborhood $Ngd(x_0, y_0, \sigma_I)$. However, if we use a larger σ_I, then the true translation (h_x, h_y) will be more likely to vary over the neighbhorhood. What can we do?

The solution is to use a "*coarse-to-fine*" approach. We compute scale spaces $I(x, y, \sigma)$ and $J(x, y, \sigma)$. For each pixel, we first estimate (h_x, h_y) at the largest scale. Then proceed iteratively to the smaller scales. See the lecture slides for a sketch of how this is done.

Note that there are two iterations going on here. There is the coarse-to-fine iteration, from large to small scales. There is also the iteration *within* each scale (recall lecture 11).