1 The Camera

✠

1. Lens and viewpoint determine perspective
2. Aperture and shutter speed determine exposure
3. Aperture and other effects determine depth of field
4. The more the aperture is open, the more light enters
5. Double the focal length: projected object size is doubled / amount of light gathered is halved.

✠ Cameras in their most primitive form are pinhole cameras. When pinhole too big - many directions averaged to blur the image. When pinhole too small - diffraction effects blur the image. Optimal size for aperture of a pinhole camera for visible light: $\sqrt{f}/28mm$. Pinhole cameras cannot maintain sharpness while allowing more light in.

✠ "Well behaved" **thin lenses**:

1. all parallel rays converge to one point on a plane located at focal length f.
2. All rays going through center are not deviated. All rays coming from points on a plane parallel to the lens are focused on another plane parallel to the lens.
3. Setting image plane at the focal length will bring points at infinity into focus. Moving it away from the focal length brings objects closer into focus. By symmetry we can also say objects at focal length require image plane to be at infinity:

$$\frac{y'}{y} = \frac{D'}{D} = (D' - f)f \qquad \text{by similar triangles}$$
$$\frac{1}{D'} + \frac{1}{D} = \frac{1}{f}$$

4. The field of view decreases when focusing on closer objects.
5. Wide angle / telephoto?

✠

1. **circle of confusion (CoC)** is an optical spot caused by a cone of light rays from a lens not coming to a perfect focus when imaging a point source. The range of object distances over which the objects appear sharp, i.e. where the CoC is less than the resolution of the sensor (max. acceptable CoC), is the **depth-of-field**.

2. **Exposure** has two main parameters:

   (a) **Shutter speed** - is the time for which the shutter exposes the sensor. It has a linear effect on *exposure*. It is usually measured in fraction of second: $1/25$, $1/60$, $1/125, 1/250, \ldots$. The blades used for shutter should be designed in a way that all of the sensor is exposed at almost the same time and for equal amount of time. Also flash needs to be in sync with the shutter.

   (b) **Aperture** - Diameter of the lens opening. It is expressed as the fraction of the focal length, in **f-number** - small f-number, big aperture:

   $$\text{f-number} = f/D$$

   F-number is adjusted in discrete steps called the **f-stop**. They are adjusted as powers of $\sqrt{2}$: $f/1$, $f/1.4$, $f/2$, $f/2.8, \ldots$. Note each f-stop halves the light intensity from the previous f-stop. *Depth-of-field* also increases with f-number (smaller aperture openings). If the aperture diameter is halved, the depth of field is doubled because of the effect on the max. possible CoC.

3. The rule of **Reciprocity** says that the same *exposure* is obtained with an exposure twice as long and an aperture area half as big. This rule can be break down for very long exposures. If we know the amount of exposure needed, the choice of shutter speed will be dictated by motion blur, camera shake; whereas the choice of aperture will be dictated by the depth of field, diffraction limit.

4. **Metering** is the brains behind how your camera determines shutter speed and aperture, based on lighting conditions. Since the camera can only measures reflected light and reflectance between objects can greatly vary, this is not a simply job. For this reason in-camera metering is standardized for the luminance of light reflected from an object appearing middle gray (reflects 18% of incident light). Hence, its quite apparent why there is a problem with objects reflecting more or less light. Metering options usually work by assigning a weighting to different light conditions for calculating exposure settings; those with a higher weighting are considered more reliable, and thus contribute more to the final exposure calculation. **Spot-Metering** only considers a small region for this calculation. Cameras come with Aperture and Shutter-speed priority modes.

5. The camera's **ISO** setting or *ISO speed* is a standard which describes its absolute sensitivity to light. ISO settings are usually listed as factors of 2, such as ISO 50, ISO 100 and ISO 200 and can have a wide range of values. Higher numbers represent greater sensitivity and the ratio of two ISO numbers represents their relative sensitivity, meaning a photo at ISO 200 will take half as long to reach the same level of exposure as one taken at ISO 100 (all other settings being equal). It is accomplished by amplifying the image signal in the camera, however this

also amplifies noise and so higher ISO speeds will produce progressively more noise.

✠ A **lens system** is there in a camera to correct for the many **optical aberrations** - they are basically aberrations when light from one point of an object does not converge into a single point after passing through the lens system. *Optical aberrations* fall into 2 categories: **monochromatic/geometrical** (caused by the geometry of the lens and occur both when light is reflected and when it is refracted - like pincushion etc.); and **chromatic** (Chromatic aberrations are caused by dispersion, the variation of a lens's refractive index with wavelength). Lenses are essentially help remove the shortfalls of a pinhole camera i.e. how to admit more light and increase the resolution of the imaging system at the same time. With a simple lens, much more light can be bought into sharp focus.

Different problems with lens systems:

1. **Spherical aberration** (Geometric) - A perfect lens focuses all incoming rays to a point on the optic axis. A real lens with spherical surfaces suffers from spherical aberration: it focuses rays more tightly if they enter it far from the optic axis (smaller focal length) than if they enter closer to the axis. It therefore does not produce a perfect focal point and results in image blurring.

2. **Astigmiatism** (Geometric) - where rays that propagate in two perpendicular planes have different foci. So the point of best focus (having max. CoC) might make horizontal and vertical lines go out of focus.

3. **Coma** (Geometric) - results in off-axis point sources such as stars appearing distorted, like having a comet shaped blob.

4. **Chromatic Aberration** (Chromatic) - rays at different wavelengths tend to have different focal points. This is directly a property of changing refractive index with wavelength of light. The usual effect is coloring at the fringe.

5. **Vignetting** (Geometric) - There are several causes of vignetting and one of them is optical vignetting which is caused by the physical dimensions of a multiple element lens. Rear elements are shaded by elements in front of them, which reduces the effective lens opening for off-axis incident light. The result is a gradual decrease in light intensity towards the image periphery.

6. **Lens Flare / Scattering** (Geometric) at the lens surface - Some light entering the lens system is reflected off each surface it encounters.

7. **Radial Distortion** (Geometric) - *Barrel* (happens in wide-angle lens) and **Pincushion** (happens in tele-photo lens) distortion.

✠ Different kind of optical sensors include **Charge coupled devices (CCD)**. It is simply an

array of **photosites** which can be thought of as buckets to hold electrical charge. The amount of charge stored is proportional to the incident light intensity during exposure. There are two methods of transferring data to ADCs: (1) interline transfer, (2) frame transfer. The digitization is done using ADCs line by line. Since the a photosite can get saturated, it can cause **blooming** in the final image. Also a CCD can produce a thermally generated image (in darkness) - which is called the **dark current**.

CMOS on the other hand has an amplifier on each photosite. It also tends to have more noise and lower sensitivity. But can put more components on the same die.

✠ A pixel is not a square (or rectangular). It is simply a point sample. There are cases where the contributions to a pixel can be modeled, in a low order way, by a little square, but not ever the pixel itself. The sampling theorem tells us that we can reconstruct a continuous entity from such a discrete entity using an appropriate reconstruction filter - for example a truncated Gaussian.

✠ Color cameras:

1. **Prism color camera** separates light into 3 beams using dichroic prism. It requires 3 sensors for capturing the separated light.
2. **Filter mosaic** is to coat filter directly onto the sensor (like a bayer pattern) - then demosaic to obtain the full color image in full resolution. This gives us our usual CCD cameras.
3. **Filter wheel** works by rotating different filters infront of the lens - but only suitable for static scenes.
4. **CMOS Foveon's X3** works by sampling all three colors at the same pixel.

✠ **White balance** is the process of removing unrealistic color casts, so that objects which appear white in person are rendered white in the image. This variation can happen under different lighting conditions (remember the color signal is a product of the illumination and reflectance).

2 Image Compositing and Blending

✠ **Compositing** is the process of digitally assembling multiple images to make a final image. Compositing by simple pixel replacement usually doesn't work especially on boundaries and transparency (shadows). Other problems include semi-transparent objects, and pixel spacing being too large.

✠ Usual solution is add one more channel - **Alpha Channel** making the image (R, G, B, α). The process of **alpha blending** uses an opacity value $\alpha \in [0, 1]$ to control the proportions of two input pixel values that end up a single output pixel. $\alpha = 0$ fully transparent; $\alpha = 1$ fully opaque:

$$I_{\text{comp}} = \alpha I_{\text{fg}} + (1 - \alpha) I_{\text{bg}}$$

A monochrome raster image where the pixel values are to be interpreted as alpha values is known as the **matte**.

Multiple images can be alpha blended together iteratively. First two images C_a and C_b are alpha blended to give C_{comp} and α_{comp}, which is used in compositing with other images:

$$C_{\text{comp}} = C_a \alpha_a + (1 - \alpha_a) C_b \alpha_b$$
$$\alpha_{\text{comp}} = \alpha_a + (1 - \alpha_a) \alpha_b$$

Of course pre-multiplying α helps. `http://en.wikipedia.org/wiki/Alpha_compositing`

✠ **Feathering** is the process of linear alpha blending across a seam. It is used to smooth or blur the edges of a feature OR as in our case combine two features/images. Important consideration here is the window size - choose a window size which smooths but does not produce "ghosts". There are multiple methods for choosing α for blending for alpha blending for panoramas:

1. Simple (constant) averaging region across the seam.
2. Computing distance transform to their respective edges for both the images - then computing:

$$\alpha = \begin{cases} 1 & \text{if } d_{1,\text{trans}} > d_{2,\text{trans}} \\ 0 & \text{otherwise} \end{cases}$$

3. Blurring alpha close to the seam
4. Center weighted alpha - best result but visible "ghosting":

$$\alpha = \frac{d_{1,\text{trans}}}{d_{1,\text{trans}} + d_{2,\text{trans}}}$$

Now the problem comes down to choosing the right window size:

$$f_{\text{largest}} \leq 2 f_{\text{smallest}}$$

where the largest and smallest frequencies correspond to object sizes. Decompose the Fourier images into octaves (frequency bands where limits are in 1:2 ratio) and combine F^i_{left} and F^i_{right} in spatial domain. This can also be seen as pyramid blending:

1. Build Laplacian pyramids L_A and L_B from the two images.
2. Build Gaussian G_r pyramid around the region you want to blend.
3. Form the combined Laplacian pyramid:
$$L_{\text{comp}}(i,j) = G_r(i,j)L_A(i,j) + (1 - G_r(i,j))L_B(i,j)$$
4. Collapse L_{comp} to make the final image.

Brown-Lowe 2003 proposed a simplification - **Two-band blending**. Which blends low frequencies smoothly and high frequencies with no smoothing (binary alpha).

✠ Rather than doing blending over Laplacian, why not over 1^{st} derivatives (gradients) - this is **Gradient Domain blending**:

1. Compute the partial derivatives dx and dy (the gradient field) for the region you want to blend.
2. Smooth / blend / feather using the derivatives.
3. Since the integrals might not be equivalent now, *reintegrate* to find an agreeable solution.
4. Reintegrate

✠ Rather than blending, we can also cut pixels. But now the problem is to make cuts that the blending itself is least visible i.e. finding the optimal seam. The problem of finding the minimum error boundary can be solved by dynamic programming. But dynamic programming can't handle loops for cuts in closed regions. To find the exact *best seam* solution for compositing, we use **Graph Cuts** with the following steps:

1. Express this minimization in terms of finding the min-cut on a graph where edge costs correspond to seam costs.
2. Solve the min-cut by solving an equivalent problem of finding the max-flow from source to sink of the graph.

3 Time-Lapse Video Analysis & Editing

✠ **Time-Lapse Photography**: when frames are captured at a much slower rate than the rate

at which they will be ultimately played back. Challenges: (1) lighting changes (strobing), (2) high frequency motion (missed action), (3) camera motion (both intentional and accidental), (4) triggering/data storage/camera safety.

✠ Bennett & McMillan 2007 **Computational Time-Lapse Video** - Uses a *Virtual Shutter* to do non-uniform sampling (with motion trails). This would technique would extract the same number of frames (the output video would be same length) but more frames are pulled from temporal locations where there is greater change in the scene. This done by using something that is called "Optimum polygonal approximation of digitized curves" (something like smart sampling without needing to always sample above the Nyquist rate). This is done by finding the global minimum with the *DP*.

✠ **Albedo** of an object is a measure of how strongly it reflects light from light sources such as the sun. Albedo is defined as the ratio of total-reflected to incident electromagnetic radiation. It is good first approximation to *BRDF*.

✠ Sunkavalli et al. 2007 **Factored Time-Lapse Video** - aims to capture outdoor scenes. It answers questions such as can we remove shadows, render new shadows, change albedo, change global illumination? So it starts by trying to separate lighting due to the sun, sky and finding the surface normals. Hence the problem is formulated as:

$$F(x,y,t) = \underbrace{I_{\text{sky}}(x,y,t)}_{\text{accumulated intensity from sky}} + \underbrace{S_{\text{sun}}(x,y,t)}_{\text{binary }|\text{in shadow}} * \underbrace{I_{\text{sun}}(x,y,t)}_{\text{accumulated intensity from sun}}$$

Note, whenever the sun's contribution is 0, there will only be sky's contribution. Use Photoshop to pick out non-surfaces so that we don't need to calculate sky's contribution for those pixels. First we calculate S_{sun}. This is done per pixel picking the median of the darkest 20% and thresholding. Run the bilateral filter to get the final S_{sun}. We can estimate the change in skylight if we consider:

$$I_{\text{sky}}(x,y,t) \approx \underbrace{W_{\text{sky}}}_{\text{sky-light image}} \times \underbrace{H_{\text{sky}}(x,y,t)}_{1D \text{ sky-light basis curve}}$$

Using *Alternating Least Squares (ACLS)* we optimize this factorization by holding $H(t)$ constant and optimizing W and vice versa. Now how to compute the contribution of the sun light:

$$I_{\text{sun}}(x,y,t) \approx \underbrace{W_{\text{sun}}}_{\text{sun-light image, per pixel weight}} \times \underbrace{H_{\text{sky}}(x,y,t+\Phi_{(x,y)})}_{1D \text{ sun-light basis curve}}$$

$\Phi_{(x,y)}$ is the shift-map, the per-pixel offset in time. We will eventually have estimations for I_{sky}, S_{sun}, and I_{sun}. To finish we also do a last ACLS to optimize for $\Phi_{(x,y)}$. We can now also estimate 1 component of the per pixel normals.

4 Carving, Warping, and Morphing

✠ **Retargeting** operators are there to resize an image/video to fit a certain screen size. *Simple retargeting operators* can be done from homogeneous scaling or by letterboxing (adding black mattes). On the other hand we can which **Content-aware retargeting operators** which retargets by taking the importance of objects into account. But no single operator works well in all cases - in fact we need a combination seam carving with cropping and scaling.

✠ Avidan & Shamir 2007 **Seam Carving for Content-Aware Image Resizing** retargets : (1) use gradient magnitude/entropy/etc as an energy function for each pixel, (2) cut horizontal or vertical paths which have the least energy.

✠ Rubinstein & Shamir 2009 **Multi-operator Media Retargeting** uses multiple operators to retarget. It uses an order preserving image to image similarity measure called *Bi-directional warping* and devises a *DP* algorithm which finds the optimal sequence of operators to maximize the similarity. Each operators adds 2 dimensions to the resizing space. The path in the resizing space defines the order and contribution of each operators in the resizing. The optimal path is found by maximizing the similarity measure between the original image and the retargeted image.

✠ Pritch et at. 2009 **Shift-Map Image Editing** proposes a shift map based approach for image retargeting, inpainting, image rearrangement and similar applications. The *shift map* is the relative shift of every pixel in the output image from the input image. The problem takes an optimal graph labeling approach, where the shift-map represents the selected label for each output pixel. Two terms are used in computing the optimal shift-map: (1) A data term which indicates constraints such as the change in image size, object rearrangement, a possible saliency map, etc. (2) A smoothness term, minimizing the new discontinuities in the output image caused by discontinuities in the shift-map. Solved by graph cut.

✠ Remember, image filtering changes the range and image warping changes the domain. **Linear transformations** have the following properties: (1) origin maps to origin, (2) lines map to lines, (3) parallel remains parallel, (4) ratios are preserved, (5) closed under multiple transformations.

Affine transformations are a combination of linear transformations and translation. In it origin does not necessarily map to origin.

Projective transformations are a combination of affine transformations and projective warps. In it parallel lines necessarily might not remain parallel plus ratios might not be preserved. Projective transformation has 8 DOF, and needs 4 corresponding points to solve.

In **forward warping** we can "splat" contribution of a pixel to multiple pixels if it doesn't fall on an integer value. Similarly in **inverse warping** we can do nearest neighbor, bilinear, bicubic, or

gaussian interpolation. Remember, *inverse mapping doesn't lead to holes while forward mapping can.*

✠ **Mosaic**: Pencil rays contains all views - can generate any synthetic view as long as same center of projection:

1. Take images by rotating camera around optical center.
2. Compute and transform second image to the first.
3. Blend to create mosaic - repeat for more images.

Note that the mapping between any two views around the optical center (given projection planes) is given by the **homography** (projective transformation):

$$p' = \mathbf{H}p$$
$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \mathbf{H} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

✠ **DLT (Direct Linear Transform Algorithm)** for computing Homography: given $\mathbf{x}_{1\cdots N}$ in the first image and $\mathbf{x}'_{1\cdots N}$ in the second image, find the homography H which best matches these points. Generally the problem is of minimization:

$$\mathrm{H} = \arg\min_{\mathrm{H}'} \sum_{i=1}^{N} ||\mathbf{x}'_i - \mathrm{H}'\mathbf{x}_i||^2$$

Since we are dealing with homogeneous coordinates, the point $\mathbf{x}_i$ will represent the same point if it is multiplied by a scale factor: $s \begin{bmatrix} \hat{x}_i \\ \hat{y}_i \\ 1 \end{bmatrix} = s\hat{\mathbf{x}}_i$. Now we can state that $s\hat{\mathbf{x}}'_i = \mathrm{H}\mathbf{x}_i$:

$$\mathrm{H}\mathbf{x}_i = \begin{bmatrix} h_1^{\mathrm{T}}\mathbf{x}_i \\ h_2^{\mathrm{T}}\mathbf{x}_i \\ h_3^{\mathrm{T}}\mathbf{x}_i \end{bmatrix} = s\hat{\mathbf{x}}'_i$$

But we can see that $s = h_3^{\mathrm{T}}\mathbf{x}_i$, hence:

$$h_3^{\mathrm{T}}\mathbf{x}_i\hat{x}_i = h_1^{\mathrm{T}}\mathbf{x}_i$$
$$h_3^{\mathrm{T}}\mathbf{x}_i\hat{y}_i = h_2^{\mathrm{T}}\mathbf{x}_i$$
$$\underbrace{\begin{bmatrix}\mathbf{x}_i^{\mathrm{T}} & 0^{\mathrm{T}} & -x_i'\mathbf{x}_i^{\mathrm{T}} \\ 0^{\mathrm{T}} & \mathbf{x}_i^{\mathrm{T}} & -y_i'\mathbf{x}_i^{\mathrm{T}}\end{bmatrix}}_{\mathrm{A}_i} \underbrace{\begin{bmatrix}h_{11}\\h_{12}\\h_{13}\\h_{21}\\h_{22}\\h_{23}\\h_{31}\\h_{32}\\h_{33}\end{bmatrix}}_{\mathrm{h}} = \begin{bmatrix}0\\0\end{bmatrix}$$

Constructing a joint matrix, we have:

$$\mathrm{Ah} = \begin{bmatrix}\mathbf{x}_1^{\mathrm{T}} & 0^{\mathrm{T}} & -x_1'\mathbf{x}_1^{\mathrm{T}} \\ 0^{\mathrm{T}} & \mathbf{x}_1^{\mathrm{T}} & -y_1'\mathbf{x}_1^{\mathrm{T}} \\ \vdots & \vdots & \vdots \\ \mathbf{x}_N^{\mathrm{T}} & 0^{\mathrm{T}} & -x_N'\mathbf{x}_N^{\mathrm{T}} \\ 0^{\mathrm{T}} & \mathbf{x}_N^{\mathrm{T}} & -y_N'\mathbf{x}_N^{\mathrm{T}}\end{bmatrix} \begin{bmatrix}h_{11}\\h_{12}\\h_{13}\\h_{21}\\h_{22}\\h_{23}\\h_{31}\\h_{32}\\h_{33}\end{bmatrix}$$

The SVD of a matrix $\mathrm{A} = \mathrm{USV}^{\mathrm{T}}$. The columns of U, $u_1, u_2, \ldots, u_m$, and columns of V, $v_1, v_2, \ldots, v_n$ are orthogonal.

If $rank(\mathrm{A}) = r$, then the column vectors of V, $v_{r+1}, v_{r+2}, \vdots, v_n$ form a basis for the null space of A. If given the SVD, this fact is useful in solving $\mathrm{A}\mathbf{x} = 0$.
Remember that orthogonal matrix are geometrically either rotation or reflection matrices. Essentially, the multiplication $\mathrm{A}\mathbf{x}$ is (1) first a rotation (or reflection) around the origin by V^{T}; (2) non-uniform scaling by the values by S and possible change in dimensions; and, finally, (3) rotation (or reflection) around the origin by U. Note, the determinant dictates the scaling of area - if determinant is s, a unit area would become s after transformation.

Atleast 4 corresponding points should be given to solve $\mathrm{Ah} = 0$ and no 3 should be collinear. One way to find $\mathrm{Ah} = 0$, i.e. the nullspace of A, is to find the SVD of A, and take the last column of V. To explain this we need to look at the SVD. Since S is in the decreasing order of singular values along the diagonal. Since we are seeking a vector for h which will be multiplied by the smallest singular value (other than 0), the last column of V would be the perfect fit.

✠ This method will still work if we change the camera center for a *planar scene* or *from far away*, since the new PP3 is a projection plane of both centers of projection.

✠ We can also create **pushbroom mosaics**. It works well on flat streets, but coming to a halt or when the road is not flat are problems. This is why setups also try to estimate 3D pose. Close and distant object can get distorted - this asks for combine perspective images with pushbroom images.

Rom'an et al. 2004 **Interactive Design of Multi-Perspective Images for Visualizing Urban**

Landscapes, describe an interactive system for creating multi-perspective images composed of serially blended cross-slits images. Beginning with a sideways-looking video of the scene as might be captured from a moving vehicle, they allow the user to interactively specify a set of cross-slits cameras, possibly with gaps between them. In each camera, one of the slits is defined to be the camera path, which is typically horizontal, and the user is left to choose the second slit, which is typically vertical. The system then generates intermediate views between these cameras using an interpolation scheme, to produce multi-perspective image with no seams.

✠ **Local (non-parametric) image warping** lets you specify warps individually for each point. Once we specified a few corresponding points we can define a triangular mesh over the points. Now we need warp triangles to triangles which can be done using an affine warp (or simply using barycentric coordinates). Now we can **cross-dissolve** images to create a video where $t \in [0, 1]$ and doing linear interpolation of each corresponding feature point:

$$(1 - t)p'_i + tp_i$$

✠ **View Morphing** is a technique to create interpolated views given a few views of the same object. This is done by: (1) pre-warp given images to form parallel views with a homography, (2) generate the novel by interpolating the pre-warped images, (3) post-warp the novel view.

5 High Dynamic Range Imaging and Tone Mapping

✠ **High Dynamic Range** images are generated by:

1. Select a range for capturing and photograph it with different exposures i.e. sequentially measure all segments of the range into a low dynamic range photograph. 4 methods to change exposure: (1) shutter speed (6 orders of magnitude, linear response but noise at long exposure), (2) aperture (2.5 orders of magnitude but depth of field changes), (3) ISO (1.5 orders of magnitude but noise at high ISO), (4) **neutral density filter** modifies intensity of all wavelengths of light equally (4 orders of magnitude, works with flash, but have to touch camera and not perfectly neutral - color shift).

2. Estimate a **camera response curve** (how actual brightness maps to pixel values from the camera's sensor). This can be done by varying the exposure and measuring values from a large number of pixels (Δt is the exposure time):

$$\begin{aligned} \text{Pixel Value } Z &= f(\text{Exposure}) \\ \log \text{Exposure} &= \log \text{Radiance} + \log \Delta t \end{aligned} \tag{1}$$

Now we will have different graphs for Z against log Exposure for each individual pixel if we suppose unit radiance for each pixel. We need to adjust the radiances to obtain a smooth response curves. Let us suppose $g(Z)$ is the discrete inverse response function and we want to solve for each pixel site i in each image j:

$$\log \text{Radiance}_i + \log \Delta t = g(Z_{ij})$$

This is solved as an overdetermined linear system. Once we have solved this, given a pixel value we can get the log Exposure. Note we will have different response curves for the 3 channels.

3. Now we can compute the radiance using Equation 1 - which will result in the whole **radiance map** of the image.

✠ How to do image registration with images of different exposure:

1. Use grayscale versions of the photographs
2. Threshold at the image at the median - **Median Threshold Bitmap (MTB)**.
3. Find the translation that minimizes the difference, which can be accelerated using pyramids.

✠ **HDR Video** can be done by taking images from the video camera with two alternating exposure settings. They help combine to more dynamic range video.

✠ **Radiometric Camera Calibration** robustly estimate cameras photometric response curve, exposure, white balance changes, and vignetting of moving cameras.

✠ HDR images can be encoded into: (1) Adobe DNG (digital negative), (2) RGBE (RGB + 8 bit exponent), (3) OpenEXR (16 bit per channel, 10 bit mantisa, 5 bit exponent) (4) JPEG 2000 (lossy 16 bit per channel)

✠ Humans deal with dynamic range by having multiplicative (in ratios) contrast sensitivity. Plus in the pupil, neural and chemical changes help in dynamic adaptation. But we have different contrast sensitivity to different frequencies - the contrast sensitivity function tells us we have highest contrast sensitivity of medium to high frequencies. Most methods that deal with tone mapping reduce the contrast of low frequencies.

✠ One way of tone mapping is to do **gamma compression** $X \to X^{0.5}$. But this washes out colors. If we simply do this on intensity colors become alright but high frequency detail is blurred. We can also just reduce the contrast of low frequencies and keep the high frequencies. This induces halos especially at strong edges.

6 Bilateral Filter

✠ Normalized Gaussian blur can be written in this notation:

$$\underbrace{GB[I]_p}_{\text{result at pixel } p} = \underbrace{\sum_{q \in S}}_{\text{sum over gaussian's space}} \underbrace{G_\sigma(\| p - q \|)}_{\text{normalized Gaussian function}} I_q$$

Note that Gaussian filter blurs across edges; value on the filter is a only function of the spatial distance from the center.

✠ The **Bilateral Filter** is nonlinear filter which improves on the Gaussian by using a kernel which is reminiscent of the image content below:

$$BF[I]_p = \underbrace{\frac{1}{W_p}}_{\text{normalization factor}} \sum_{q \in S} \underbrace{G_{\sigma_s}(\| p - q \|)}_{\text{normalized Gaussian function}} \underbrace{G_{\sigma_r}(|I_p - I_q|)}_{\text{range weight}} I_q$$

Now σ_s encodes the extent of the kernel, size of the considered neighborhood; σ_r encodes the minimum amplitude of an edge. Hence only pixels close in space and range are considered. σ_r can be set according to the mean or median of image gradients.

✠ The applications of *Bilateral Filter*:

1. Image denoising: a small window size for σ_s and σ_r is set according to the expected noise level.
2. **Tone mapping**: it used for contrast reduction for low frequencies. Using the log domain for range with σ_s of 2% image size and $\sigma_r = 0.4$ in log_{10} domain:
   (a) Blur the intensity image with bilateral filter (where the range weight is done in log domain). This gives the large scale image.
   (b) Subtract the log intensity image by the large scale image to give the detailed image.
   (c) Reduce contrast (gamma correction) on the large scale image:

$$\begin{aligned} \text{largeRange} &= \max\{\text{inLogLarge}\} - \min\{\text{inLogLarge}\} \\ k &= \log_{10}(10)/\text{largeRange} \\ \text{outLog} &= \text{inLogDetail} + \text{inLogLarge} \times k - \max\{\text{inLogLarge}\} \end{aligned}$$

   (d) Combine the color, detail, and large scale image to give the output HDR.

3. Flash / No-Flash Photography: how to incorporate the natural light in the scene while keeping the detail visible from flash photography. Use a **cross-bilateral filter** which does spatial smoothing in image A which is weighted by intensity similarities in image B. Use $CBF(A, B)$ to remove A's noisy details; use $CBF(B, A)$ to remove B's less noisy details; add them together to get clean sharp image.

7 Re(Coloring)

✠ **Colorization** is a computer-assisted process of adding color to a monochrome image or movie.

✠ Levin et al. 2004 **Colorization Using Optimization**: Using a few user provided scribbles, the color is propagated to both space and time neighboring pixels with similar intensity values. Thinking of this in the YUV space, we already have the intensity Y. To get U and V we will solve the following minimization problem. This is done by minimizing the difference between color at a pixel an affinity weighted average of neighbors:

$$J(U) = \sum_r \left(U(r) - \sum_{s \in N(r)} w_{rs} U(s) \right)^2$$
$$w_{rs} \propto e^{-(Y(r)-Y(s))^2/\sigma_r^2}$$

where w_{rs} is the affinity function where σ_r is proportional to the local variance. Since the cost is quadratic, minimum can be found using a sparse system of linear equations.

In recoloring, the affinity between pixels can be based on intensity and *color similarities*.

✠ F. Piti, 2005 **N-Dimensional Probability Density Function Transfer and its Application to Colour Transfer** proposes an original method to estimate a continuous transformation that maps one N-dimensional distribution to another. The method is iterative, non-linear, and is shown to converge. Only 1D marginal distribution is used in the estimation process, hence involving low computation costs.

8 Texture Synthesis

✠ Efros & Lueng 1999 **Texture Synthesis by Non-Parametric Sampling** - uses Markov property to compute probability density function for synthesizing each pixel at a given location $P(p|N(p))$ i.e. it search for similar neighborhoods compared to the current location. To sample from the PDF,

we just pick a match at random. Details: (1) Pixels with most neighbors are synthesized first, (2) If no match found, delay its synthesization to the end, (3) Uses Gaussian weighted SSD to match, (4) Neighborhood window size an important factor.

✠ Efros & Freeman **Image Quilting** uses the idea patch based synthesis, since neighbors are highly correlated. Now we will compute the pdf for a block $P(B|N(B))$. Each block is synthesized starting from the top left corner going right and down. Each block is placed with an overlap with the previous blocks, and the block is chosen in a way that the SSD in the overlap is minimized. Afterwards the minimum error boundary is found by *DP*.

This work also does *texture transfer*. This uses the image which need to be synthesized with the new texture as a constraint. So this follows that we need to maintain the consistency of texture that is being synthesized plus their should be similarity with the underlying image.

✠ Hertzmann et al. **Image Analogies** work also synthesizes textures in scan line order. But tackles a slightly different problem. Given an image A and its filtered output A' (could be any user chosen filter), we try to construct to find what would be the output B' using this filter on B. They use this for texture synthesis, super-resolution, texture transfer, traditional and artistic filters. The synthesis is done on a Gaussian pyramid going from the coarsest to the finest level l. When synthesizing pixel q in B'_l, it looks at the neighborhood of the corresponding pixel in B'_l, B'_{l-1}, B_l, and B_{l-1}. Also it searches for the closest matching pixel by consulting A'_l, A'_{l-1}, A_l, and A_{l-1}, and then assigns the pixel from A'_l. The matching is done by using a set of feature vectors, which includes RGB.

✠ Hays & Efros **Scene Completion Using Millions of Photographs** is a method to patch up the holes in an image by finding similar image regions from the database of photographs. This is a completely data driven approach with no labeling required from the user. (1) the scene descriptors for the input image are generated, (2) these scene descriptors are then used to find matches in the database of photographs, (3) once appropriate matches are found, example outputs are generated by using context matching and blending techniques using texture and color. It uses gist scene descriptors which aggregates oriented edge responses at multiple scales into very coarse spatial bins. They also add color information of the query image downsampled to the spatial resolution of the gist. They then compute the SSD of the gist between images for matching.

9 Rendering a Scene from a Single Photograph

✠ **Image-based modeling and rendering (IBMR)** methods rely on a set of 2D images of a scene to generate a 3D model and then render some novel views of this scene. These methods include view morphing, panoramic imaging, etc.

✠ Chen & Williams 1993 **View-interpolation for image synthesis** uses interpolation between images portray 3D scenes. Uses morphing (which combines interpolation of texture maps and their shape) to compute arbitrary intermediate frames from a set of input images. If the images are structured set of views of a 3D object or a scene, intermediate frames derived from morphing can be used to approximate intermediate 3D transformations of the object or scene. The paper argues for two advantages: (1) 3D representation of the scene can be replaced with the images, (2) image synthesis time is independent of scene complexity. The pairwise correspondence between two successive images can be pre-computed and stored as a pair of *morph maps*. Using these maps, corresponding pixels are interpolated interactively under the user's control to create in-be- tween images. To speed up the computation, they use quadtrees i.e. when depth changes a-lot, we increase the resolution.

✠ Horry et al. 1997 **Tour into the Picture** aims to create cut-out walk-throughs in images. It models the scenes as a set of planes whose orientation needs to be found out. It exploits the fact that parallel lines merge at a vanishing point, and the vanishing points of different parallel lines on the same plane define a vanishing line. Plus different planes will have different vanishing lines. This gives us a way to compute the height of an object given a reference height in the scene. The idea:

1. Create a theater stage of 5 planes (the assumption is that the walls of this volume are orthogonal, camera view is parallel to rear wall, and camera up is normal to the floor). Using the planes we can define 3 sets of parallel lines / 2 of them ending at a vanishing point. This *vanishing point is supposed to be at eye level: ray from COP to vanishing point is perpendicular to image plane.*
2. Manually identify foreground objects through bounding polygons
3. Texture map the box walls with texture from the image
4. By just deciding the position of the vanishing point (constrained inside the rear wall) we can define a view point. (ratios of the lengths on the rear wall can be used)
5. For foreground objects, two points are already on the known plane - the two top points are estimated with similar triangles.

✠ Hoiem et al. 2005 **Automatic Photo Pop-up** takes a slightly different approach is it tried to learn the structure of the world (appearance-based models of geometry) to decide about the planes

in the scene. Use geometric cues to decide about the planes in the scene: use color, texture, location, and perspective. They consider a ground plane over which piece-wise vertical objects are popped up, and their is a sky. Method:

1. Given the input image and the learned geometric models from images.
2. Segment the image into superpixels using Felzenszwalb.
3. Decide which superpixels come from the same surface to form *constellations* - this is learned from the training images (using boosted kernel density estimation). By randomly initializing the initial labeling they generate multiple segmentations with constellations.
4. For each constellation, estimate the likelihood of each of three possible labels (done by using many weak geometric cues), and the confidence that all the superpixels in a constellation belong to the same label. Eventually the superpixels final label is inferred from the likelihoods of the constellations that hold that superpixel:

$$P(y_i = t|\mathbf{x}) = \sum_{k:s_i \in C_k} \underbrace{\underbrace{P(y_k = t|\mathbf{x}_k, C_k)}_{\text{label likelihood}} \underbrace{P(C_k|\mathbf{x}_k)}_{\text{homogeneity likelihood}}}_{\text{learned from training images}}$$

5. Estimate where the objects lie in relation to the ground by seeing where the vertical meets the ground in each case (using iterative hough transform). We also join line segments that intersect at a small angle. The horizon position is estimated from geometric features and the ground labels (in short, from perspective cues). We make verticals from large angle lines and *fold* along these *polylines*. We *cut* ends of polylines where they meet the sky.

✠ Guo et al. 2001 **Hybrid stereo camera: An IBR Approach for Synthesis of very High Resolution Stereoscopic Image Sequences** generates high quality CG and live action stereoscopic sequences. It supposes that low res. images for the views to be generated at high res. are available. The paper proposes a *Hybrid Stereo Camera* in which one view is rendered at the target high resolution but the other is rendered at a much lower resolution - and the aim of the paper is to enhanced the low res. view to match the quality and resolution of the full res. eye using IBR and image analysis techniques. Methods for synthesizing the second view sequence at high resolution are seen as generating intermediate views in a graph layered by the available high res. and low res. views. The hybrid resolution stereo images are processed as follows:

1. To generate the high res. for left low res. frame at time t, we look at the high res. right frame at time $t-2$, $t-1$, t, $t+1$, $t+2$.
2. We compute the per-pixel disparity at t
3. We estimate the flow-corrected disparity at $\pm\Delta t$

4. Synthesize and evaluate quality measure - to find low quality pixels (which are filled in by regular up-sampled pixels)

✠ Bhat et al. 2007 **Using Photographs to Enhance Videos of a Static Scene** tries to correct videos (over-exposed areas, limited res. and so on) by using higher res. photographs of the same scene (among other reasons they are better exposed). Essentially we want to transfer proper exposure (and high res.) from photographs to the video. They also tackle the problem of video editing - by just manually editing a photograph of the scene (or the frame itself) which is eventually propagated throughout the video. The method:

1. The system is input the low-quality video, the high-res photographs and optionally edited photographs.

2. Using the inputs, the geometry of the scene is estimated (SFM (used as a soft-prior)→ multi-view stereo (with non-fronto parallel planes)) to give the depth maps for the whole video sequence, and also the depth maps for the images. To do this, first, the scene geometry is estimated by recovering a sparse 3D point cloud of the scene - to get the depth map and the relative poses of the video frames and still photographs. This is based on multi-view stereo work by Zitnick et al. 2004, which uses an over-segmentation based approach to maintain depth consistency across views.

3. *Video-Based Rendering*: Using the depth maps, it reconstructs an enhanced video using the appearance from photographs. The spatial appearance is created while preserving the temporal dynamics of the video such as highlights and dynamic lighting. This is done by using the depths computed from the previous phase to warp the photographs into the viewpoints of the video frames. The input video is reconstructed from these warped photographs (projected into the scene) where the cuts are made using graph cuts (this is essentially reconstructing video frames from near-by photos). The data term in graph cuts encourages (1) depth matching, (2) color matching, (3) no holes. The smoothness term encourages (1) seamless label transition, (2) seams to run through regions of similar color OR along strong edges.

4. The reconstruction can have holes, spatial seams, loss of dynamic lighting effects and other temporal incoherences - this is corrected by *Space-time fusion* which is done through the 3D gradient field of enhanced video (gradient field integration is combining g_x and g_y to give the original image) which incorporates the temporal gradients of the input video (*gradient domain compositing*) in a way that they are compatible with spatial gradients:

   $$G_t(x, y, t) = V_f(x, y, t) - V_f(x - u, y - v, t - 1)$$

   This is done by solving a linear system. This helps fill regions across seams (using adjacent photographs), holes (using the video frame) and seams across holes (using video frame and the adjacent photograph).

This technique can be used to generate a HDR video by combining exposure bracketed photographs

with the LDR video. Also filters can be applied, objects can be removed (in-painting), and camera shake can be adjusted using the proposed techniques.

10 Video Texture Synthesis

✠ The Markov chain is defined as the probability of going from state i to state j in n steps is given by:

$$\Pr{}_{ij}^{(n)} = \Pr(X_1 = j | X_0 = i) \Pr(X_0 = i)$$
$$\Pr{}_{ij}^{(n)} = \sum_{r \in S} \Pr{}_{ir}^{(k)} \Pr{}_{rj}^{(n-k)} \qquad \textit{Chapman-Kolomogorov equation}$$

The Markov assumption says that the starting state of the chain has little or no effect on $\Pr(X)$ after many steps.

✠ Schodl et al. 2000 **Video Textures** aims to create loop over videos (textures) with a limited time-sequence video. The main problem is to find good transitions between frames to jump over. Steps:

1. Compute the L2 distance matrix: $D_{ij} = \| I_i - I_j \|_2$. This will help dictate good transitions can only occur across similar frames. We set the cost of a transition as: $C_{i \to j} = D_{i+1,j}$, i.e. the cost is the L2 distance between the successor and the chosen frame:

2. The probability assigned to each transition is:

$$P_{i \to j} \approx \exp(-C_{i \to j} / \sigma^2) \tag{2}$$

   where σ dictates the acceptable cost for a transition.

3. To preserve dynamics, the cost matrix is constructed by assigning diagonal weights to encourage forward transitions:

$$C'_{i \to j} = \sum_{k=-N}^{N-1} w_k D_{i+k+1, j+k}$$

4. To avoid dead-ends we propagate future transition costs backward. This is done by iteratively computing a prediction of the increase in "future cost":

$$F_{i \to j} = C'_{i \to j} + \alpha \min_k F_{j \to k}$$

   The constant α is used to control the relative weight of future transitions in the metric. For convergence, we must choose $0 < \alpha < 1$ (in practice, $0.99 \leq \alpha \leq 0.999$). This technique is known as *Q-Learning*. Now $F_{i \to j}$ replaces $C_{i \to j}$ in equation 2

5. To avoid visible jumps, we do *cross-fading*: for crossfading from sequence A to B

$$I_{i-1} = \frac{3}{4}A_{i-1} + \frac{1}{4}B_{j-2}, \quad I_i = \frac{2}{4}A_i + \frac{2}{4}B_{j-1}, \quad I_{i+1} = \frac{1}{4}A_{i+1} + \frac{3}{4}B_j$$

6. We can also morph between scenes by computing correspondences between pixels of all frames. Once we have the correspondences we can interpolate pixel position and color in the morphed frame.

We can also do a region based analysis, where we create a texture individually for each region.

✠ **Video sprites** are animations created by rearranging recorded video frames of a moving object.

✠ **Video Sprite Control** can be done by having the following augmented transition cost:

$$C_{i\to j} = \alpha \underbrace{C_{i\to j}}_{\text{similarity term}} + \beta \underbrace{\theta}_{\text{control term}}$$

where θ is the angle between the wanted control (mouse pointer) and the inherent velocity of the sprite. We can precompute the future cost $F_{i\to j}$ for a few angles and switch between these angles, on-the-fly, as the user requires.

✠ Kovar et al. 2002 **Motion Graphs** tries the video texture technique on Mocap data - given a user specified path, it generates a path from the Mocap data for the sprite to follow. It first generates a distance matrix like before but now its a minimization function over the transformation which takes from one Mocap frame to the next:

$$D(\mathcal{A}_i, \mathcal{B}_i) = \min_{\theta, x_0, z_0} \sum_i w_i \parallel p_i - T_{(\theta, x_0, z_0)} p_i' \parallel^2$$

✠ Schodl et al. 2002 **Controlled animation of video sprites** is an optimization algorithm for video sprites to animate realistic looking characters. It finds good frame arrangements based on repeated partial replacements of the sequence. The objective is to optimize animations w.r.t these user-defined costs:

1. Extract sprites using chromakeying
2. Compute a similar distance matrix, and find the transitions by comparing all pairs of frames - also list of training pairs.
3. Find the sequence of frames $s_1, \ldots, s_n$ that show the desired animation (constraints e.g. motion

trajectory). We are trying to optimize:

$$C(S) = \underbrace{C_s(S)}_{\sum_{i=1}^{n-1} C_{s_i \to s_{i+1}}} + \underbrace{C_c(S)}_{\text{control cost function}}$$

Rather than doing Q-learning, we must iterate to optimize frame sequences for multiple sprites jointly - precompute forward and backward costs based only on smoothness, every time a subsequence is chosen for replacement. For replacement we choose a random section to clip, and the precomputed cost paths for the visible frames (s_k and s_l) is used to get the subsequence which improves the total cost (checking among $t \in F(s_k) \cap B(s_l)$)

11 Deblurring and Deconvolution

✠ Types of blur: (1) camera shake, (2) scene motion, (3) defocus blur. Blur can be seen as the output from a sharp image after being convolved by a blur kernel which incodes the path of blurring. So this can be seen as blind and non-blind deconvolution:

1. Even **non-blind deconvolution** is not simple, since deconvolution is an ill-posed problem. If we look in the frequency domain (remember convolution is multiplication in frequency domain) the output signal is zero wherever the kernel is zero - making our efforts to recover the original signal go in vain. But we also know that natural images have sparse gradients. Hence we can put a penalty on the gradients:

$$\hat{x} = \arg\min_x \underbrace{|f * x - y|^2}_{\text{convolution error}} + \lambda \underbrace{\sum_i \rho(\nabla x_i)}_{\text{derivatives prior}}$$

   where $\rho(\nabla x) = \| \nabla x \|^\alpha$, and $\alpha = 2$ is a gaussian prior ("spread" gradients) and $\alpha = 0.8$ is a sparse prior (which "localizes" gradients)

2. **blind deconvolution** supposing linear convolution - Fergus et al. **Removing camera shake from a single photograph** notes that blurry images have different statistics than natural images (histogram of *log* pixel against gradient for blurry images don't have a range as big as natural images). Hence we can build a parametric model which tries to match the natural/sharp image statistics. So now we have three sources of information (1) blurry input image, (2) a blur prior (to incorporate our knowledge on camera shake), (3) image prior (our learned information on distribution of gradients in natural images):

$$\underbrace{\Pr(b, x|y)}_{\text{Posterior}} = k \underbrace{\Pr(y|b, x)}_{\text{likelihood (reconstruction constraint)}} \underbrace{\Pr(x)}_{\text{image prior}} \underbrace{\Pr(b)}_{\text{blur prior}}$$

Use variational Baye's approach it iteratively approximates the kernel. Note that the blur kernels were estimated from large image patches, and *not* from the input image.

✠ **Fluttered Shutter Camera** uses a ferroelectric shutter in front of the lens which is turned opaque or transparent in rapid binary succession. It can allow us to take photographs with our choice of convolution - i.e. it allows us to make a filter which doesn't touch zero and hence high frequencies are preserved. Hence the inverse of the filter will be stable. This flutter shutter exposure pattern is known as *Coded Exposure*.

12 Image and Depth in Coded Aperture Imaging

✠ The aim is to get both the *depth map* and an *all-focused image*. Note that when objects are further away from the focal plane, the point spread function increases. One way to deconvolve, as mentioned above, is to use a prior on natural images (which have visibly sparse gradients). We can also think of defocus as a local convolution with calibrated blur kernels at different depths (by knowing how the point spread function changes at different depths). Now the aim would be to find the best scale for the blur kernel, so we can deconvolve local windows with the right kernel. But the problem here is that the result of the smaller scale is not so different from correct scale filter.

The second solution is to use a special pattern in the aperture (**coded aperture**) which produces different defocus patterns from natural images and which are easier discriminate. It is important to note that the response of this coded aperture at the focal plane for the point spread function is the same as before. Coded aperture helps reduce uncertainty in scale identification (now smaller scale response is different from the correct scale response). This helps produce an all-focused image. Once we know the right scale for the filter, we can also coarsely estimate the depth of the object. Although one shortfall is the loss of light. Also note we have *essentially increased the depth-of-field by deconvolution.*

One application of this is digital refocusing from a single image.

13 Motion Capture

✠ An important concept in pose animation is **keyframing**. A key frame just defines the starting and ending points of any smooth transition.

✠ Motion capture technologies where the main aim is to do 3D triangulation on markers. Usually

systems produce data for 3 DOF for each marker, whereas the rotation has to be inferred from the relative orientation from of 3 or more markers:

1. **Active markers** (like magnetic markers) use broadcasting or sensing for localization. For this they need cable OR transmitter for power + signal. They usually are associated with unique IDs. Magnetic systems report rotation + translation.
2. **Passive markers** (like Optical markers) use retroreflective material to reflect light back that is generated near the camera lens. But since they are passive, the problem of correspondence is abound. For this they vary sizes. But passive markers only provide translation. Usually markers are placed in offset locations (asymmetric placement) for easier localization.

The choice of placement of cameras involves choosing a *wide baseline* for better triangulation v.s. *narrow baseline* for easier correspondences and predictable occlusions.

✠ Most human joints are approximated by ball joints (3 DOF); while the root (taken as the torso) has 3 translational DOF.

✠ The **inverse kinematics** problem is simply stated as, "Given the desired position of the robot's hand, what must be the angles at all of the robot's joints?" (humans do this all the time) This is in contrast to the **forward kinematics** problem, which is, "Given the angles at all of the robot's joints, what is the position of the hand?"

Forward kinematics encodes all DOF in 1 vector:

$$\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_M]$$

If we want to find the **end effector** position in 3D $\mathbf{e} = [e_1, e_2, e_3]$, our problem will take this form:

$$\mathbf{e} = f(\boldsymbol{\theta})$$

For *inverse kinematics*, we would have the end effector position $\mathbf{e} = [e_1, e_2, \ldots, e_N]$ and we want to find $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_M]$, our formulation will take this form:

$$\boldsymbol{\theta} = f^{-1}(\mathbf{e})$$

Doing *inverse kinematics* is usually perceived as an optimization of the Jacobian: linear approximation of $f(.)$:

$$J = \frac{d\mathbf{e}}{\boldsymbol{\theta}}$$

14 Intrinsic Images

✠ Reminder: A **Lambertian surface** has exhibits Lambertian reflectance: light falling on it is scattered such that the apparent brightness of the surface to an observer is the same regardless of the observer's angle of view. More technically, the surface luminance is isotropic.

✠ **Intrinsic Images** is the decomposition of an image into lighting/shading and reflectance component.

Two of the most important characteristics of the scene are its shading and reflectance. The **shading** of a scene is the interaction of the surfaces in the scene and the illumination. The **reflectance** of the scene describes how each point reflects light.

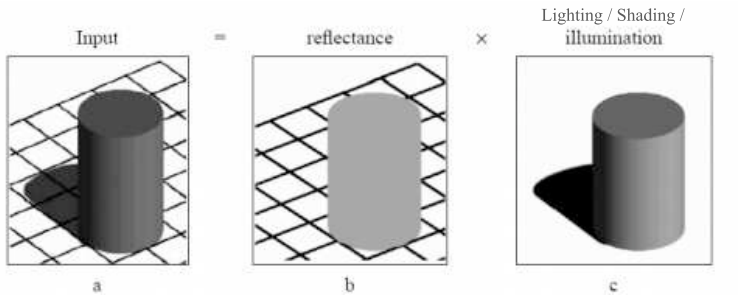

Figure 1: Decomposition into intrinsic images

✠ Sinha & Adelson 1993 **Painted Polyhedra** uses edge junctions as queue to what are illumination edges (Y and Psi junctions) and what are reflectance (T junctions) (Junction catalogue). To make the solution globally optimal we check for (1) low variance of angles, (2) planarity of faces, (3) overall compactness, (4) consistency with light source. Now there would be a possibility of the global analysis over-riding the local analysis.

✠ Weiss 2001 **Image Sequences** tackles static objects across multiple frames. The problem is formulated as given a sequence of T images $\{I(x,y,t)\}_{t=1}^{\mathrm{T}}$ in which the reflectance is constant over time and only the illumination changes, we need to find the single reflectance image $R(x,y)$ and T illumination images $\{L(x,y,t)\}_{t=1}^{\mathrm{T}}$:

$$\{I(x,y,t)\}_{t=1}^{\mathrm{T}} \;=\; \{L(x,y,t)\}_{t=1}^{\mathrm{T}} R(x,y)$$

This is still an under-constrained problem ($T+1$ unknowns and T equations. Moving to log space, and applying derivative filters can help because over time, since the illumination is changing, the derivatives across illumination edges will also move. Hence we can take the median of the response of derivative filters across time. Doing this with multiple filters will expose the reflectance image.

✠ Tappen et al. 2003 **Single Image** takes image derivatives dx and dy. We know that each derivative (edge) is either caused by shading or reflectance. Hence this boils down to binary classification problem. Steps:

1. Find image derivatives
2. Classify each derivative as caused by shading or reflectance. It is important to note that changes due to shading should effect all the 3 color channels proportionally:

   $$C_1 = \alpha.C_2$$

   if $C_1 \neq \alpha.C_2$, then the changes are caused by reflectance. Using statistical regularities of surfaces we can avoid misclassification of reflectance features as shading. Moreover we know that derivatives that lie on the same contour should have the same classification - here we propagate evidence from conclusive areas to ambiguous ones using MRF.
3. Invert derivatives classified as shading to find shading image.
4. Invert derivatives classified as reflectance to find reflectance image.

✠ Finlayson et al. 2004 **Entropy Minimization** supposes that sensors have narrow band responses (non-overlapping responses $S_k(\lambda) = \delta(\lambda - \lambda_k)$) to boil down sensor response of a pixel to just reflectance and illumination parameter:

$$p_k = \int_\lambda R(\lambda)L(\lambda)S_k(\lambda)d\lambda$$
$$p_k = R(\lambda_k)L(\lambda_k)$$

Now the important fact is that the chromaticity for single surface under different illuminants lies on a single line in $\log(B/R)$ against $\log(G/R)$ graph. So a correct projection in this space to form a grayscale image (our reflectance image) would form a tight distribution. Hence the projection direction that minimizes the entropy (more "spread-out" distribution, higher entropy), is the right projection direction.

This can only remove hard shadows and shadows shouldn't overlap with objects. Plus reconstruction methods is texture dumb.