Last update: May 4, 2010

Vision

CMSC 421: Chapter 24

Outline

- Perception generally
- Image formation
- Early vision
- 2D $\longrightarrow$ 3D
- Object recognition

Perception generally

Stimulus (percept) S, World W

$$S = g(W)$$

E.g., $g =$ "graphics." Can we do vision as inverse graphics?

$$W = g^{-1}(S)$$

Perception generally

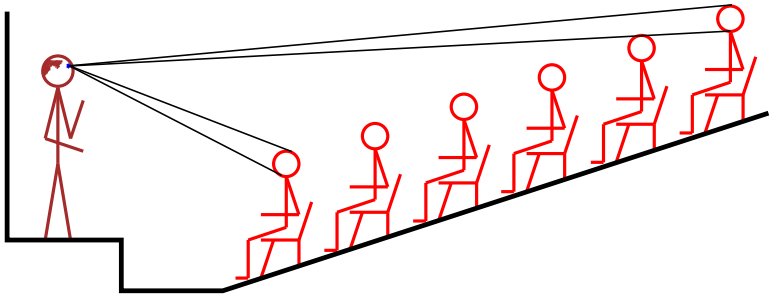
Stimulus (percept) S, World W

$$S = g(W)$$

E.g., $g =$ "graphics." Can we do vision as inverse graphics?

$$W = g^{-1}(S)$$

Problem: massive ambiguity!



Perception generally

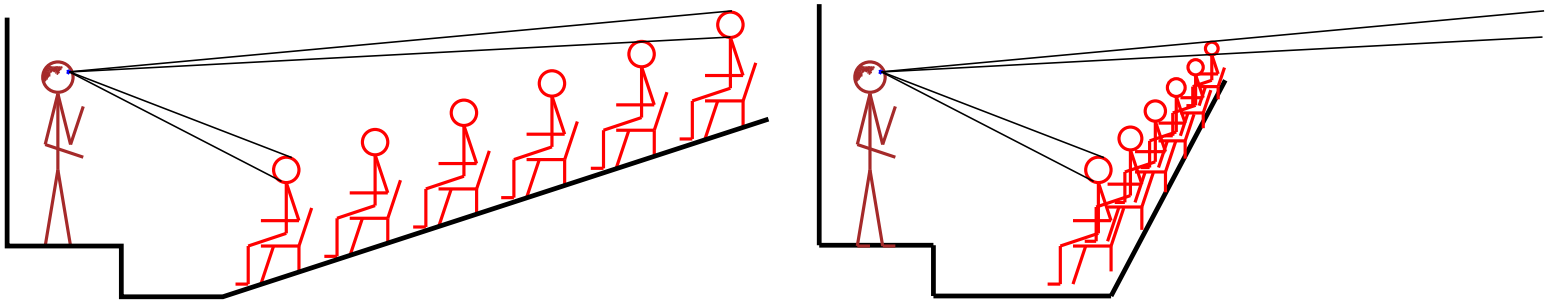
Stimulus (percept) S, World W

$$S = g(W)$$

E.g., $g =$ "graphics." Can we do vision as inverse graphics?

$$W = g^{-1}(S)$$

Problem: massive ambiguity!



Perception generally

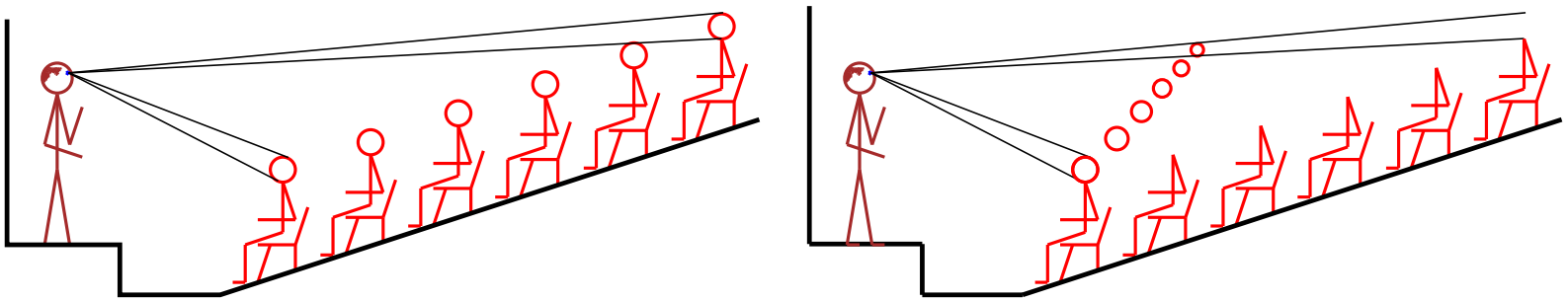
Stimulus (percept) S, World W

$$S = g(W)$$

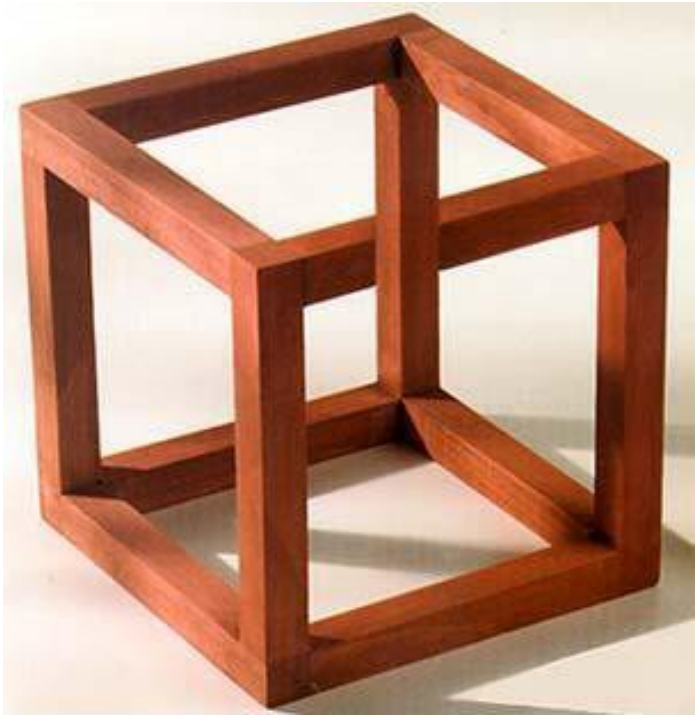
E.g., $g =$ "graphics." Can we do vision as inverse graphics?

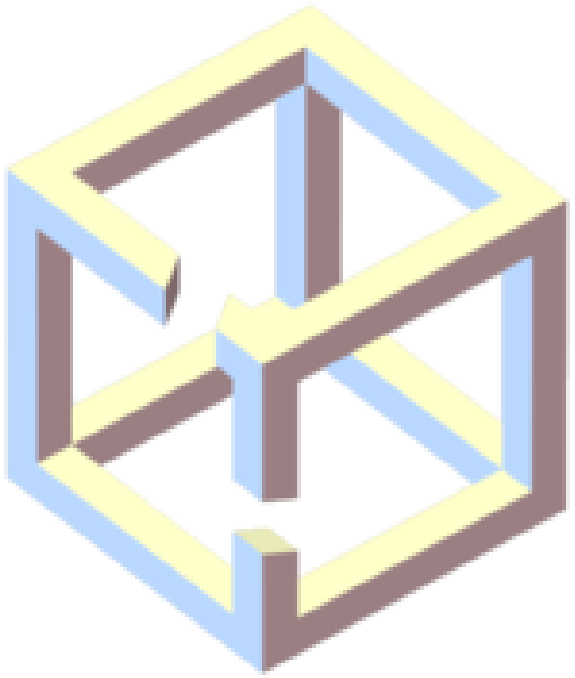
$$W = g^{-1}(S)$$

Problem: massive ambiguity!

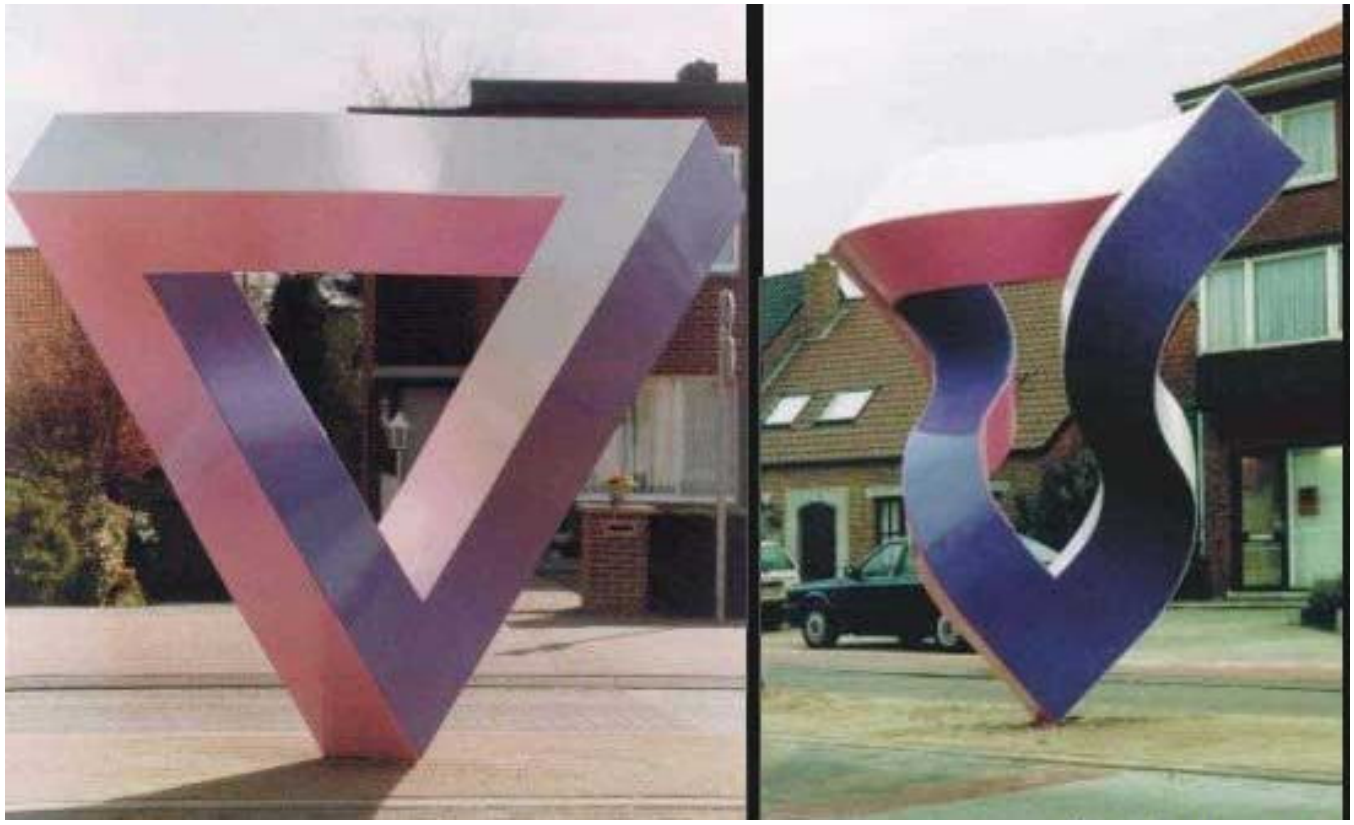


Example





Example



Better approaches

Bayesian inference of world configurations:

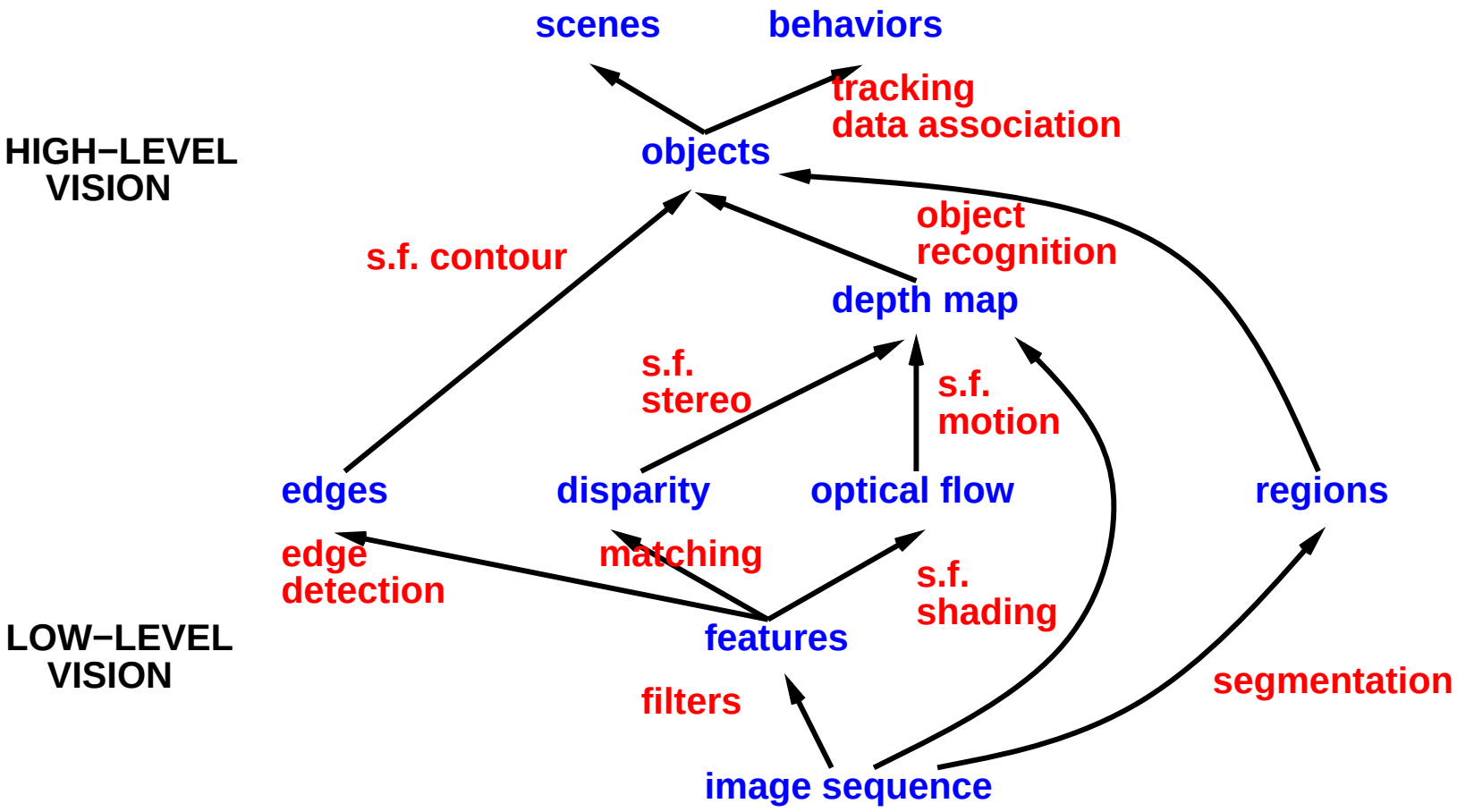
$$P(W|S) = \alpha \underbrace{P(S|W)}_{\text{“graphics”}} \underbrace{P(W)}_{\text{“prior knowledge”}}$$

Better still: no need to recover exact scene!

Just extract information needed for

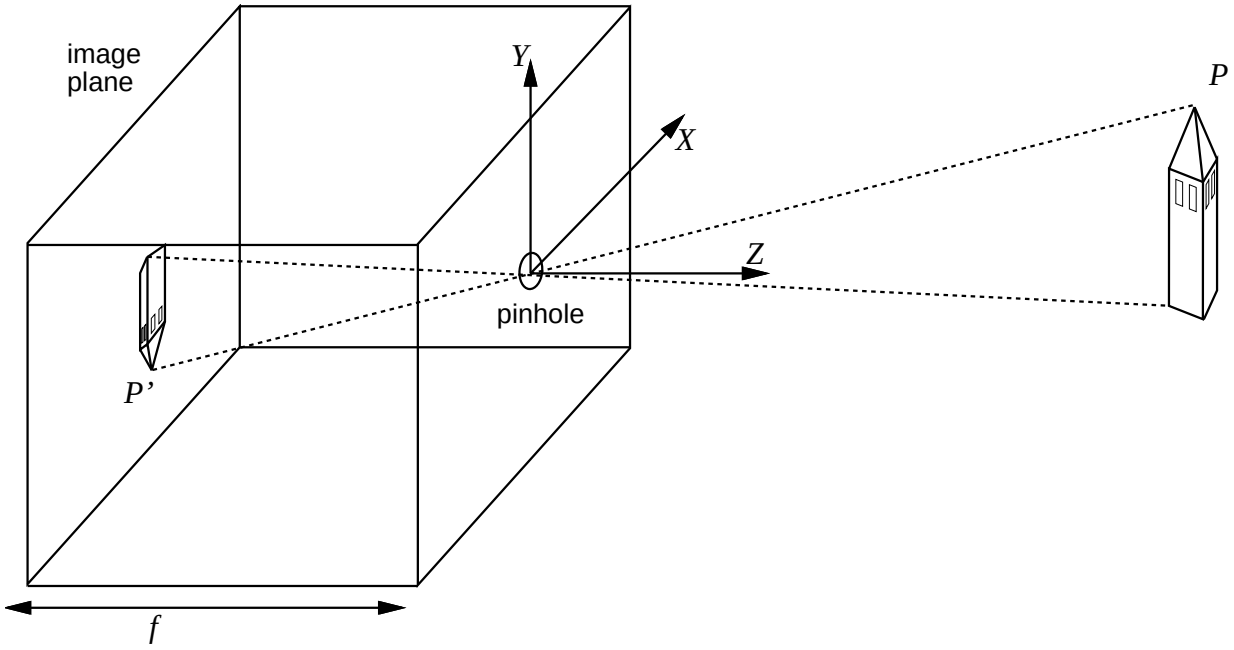
- *navigation*
- *manipulation*
- *recognition/identification*

Vision "subsystems"



Vision requires combining multiple cues

Image formation



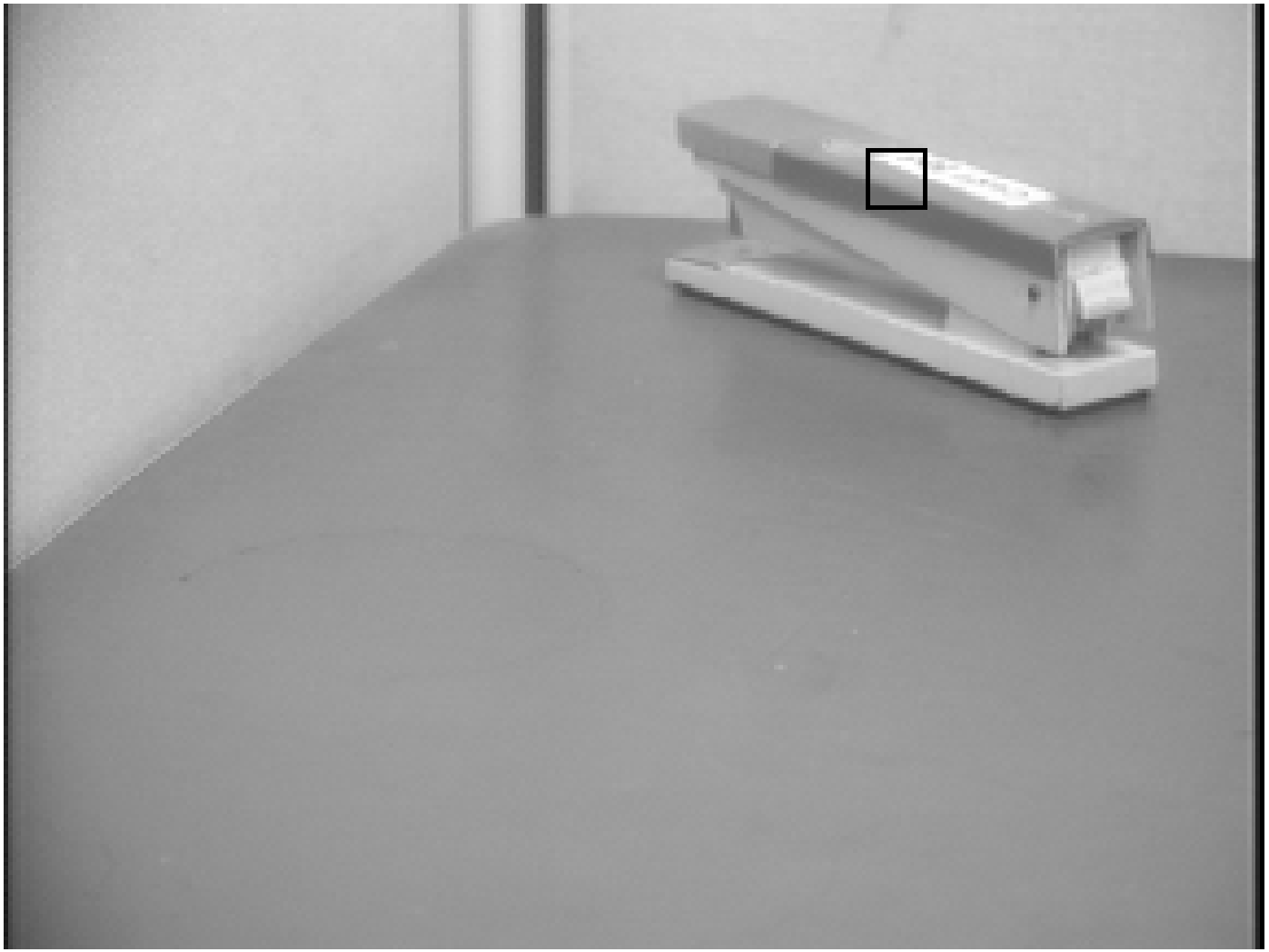
P is a point in the scene, with coordinates (X, Y, Z)

P' is its image on the image plane, with coordinates (x, y, z)

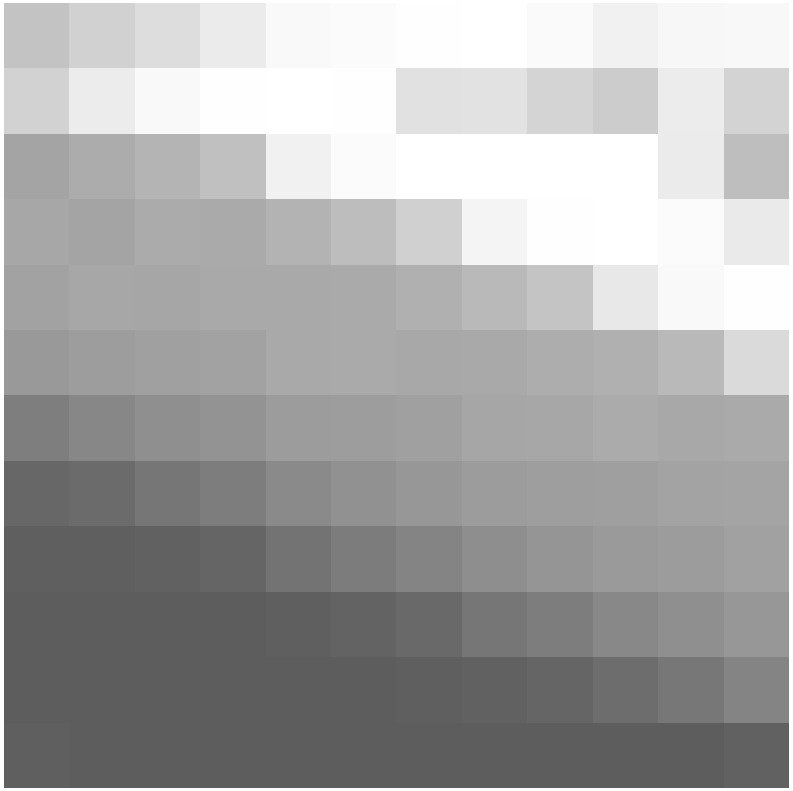
$$x = \frac{-fX}{Z}, \; y = \frac{-fY}{Z}$$

where f is a scaling factor.

Images



Images, continued



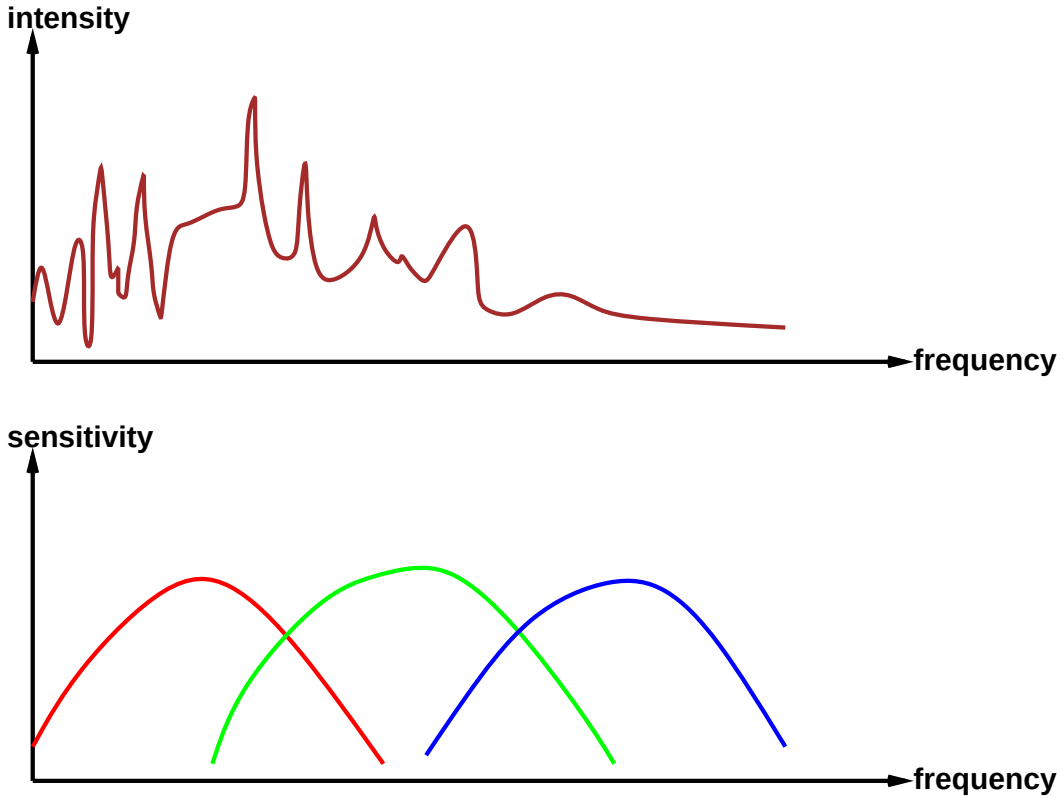
195	209	221	235	249	251	254	255	250	241	247	248
210	236	249	254	255	254	225	226	212	204	236	211
164	172	180	192	241	251	255	255	255	255	235	190
167	164	171	170	179	189	208	244	254	255	251	234
162	167	166	169	169	170	176	185	196	232	249	254
153	157	160	162	169	170	168	169	171	176	185	218
126	135	143	147	156	157	160	166	167	171	168	170
103	107	118	125	133	145	151	156	158	159	163	164
095	095	097	101	115	124	132	142	117	122	124	161
093	093	093	093	095	099	105	118	125	135	143	119
093	093	093	093	093	093	095	097	101	109	119	132
095	093	093	093	093	093	093	093	093	093	093	119

$I(x, y, t)$ is the intensity at (x, y) at time t

typical digital camera $\approx$ 5 to 10 million pixels
human eyes $\approx$ 240 million pixels; about 0.25 terabits/sec

Color vision

Intensity varies with frequency $\rightarrow$ infinite-dimensional signal



Human eye has three types of color-sensitive cells;
each integrates the signal $\Rightarrow$ 3-element vector intensity

Primary colors

Did anyone ever tell you that the three primary colors are red, yellow, and blue?

Primary colors

Did anyone ever tell you that the three primary colors are red, yellow, and blue?

Not correct.

Did anyone ever tell you that

- red, green, and blue are the primary colors of light, and
- red, yellow, and blue are the primary pigments?

Primary colors

Did anyone ever tell you that the three primary colors are red, yellow, and blue?

Not correct.

Did anyone ever tell you that

- red, green, and blue are the primary colors of light, and
- red, yellow, and blue are the primary pigments?

Only partially correct.

What do we mean by "primary colors"?

Primary colors

Did anyone ever tell you that the three primary colors are red, yellow, and blue?

They weren't correct.

Did anyone ever tell you that

- red, green, and blue are the primary colors of light, and
- red, yellow, and blue are the primary pigments?

Only partially correct.

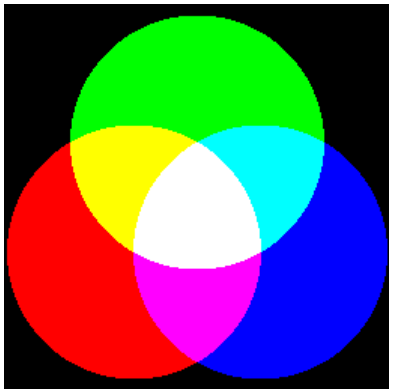
What do we mean by "primary colors"?

Ideally, a small set of colors from which we
can generate every color the eye can see

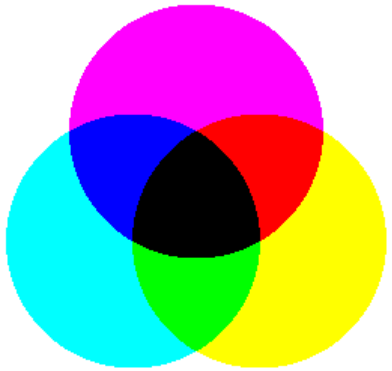
Some colors can't be generated by any combination of red, yellow, and blue pigment.

Primary colors

Use color frequencies that stimulate the three color receptors



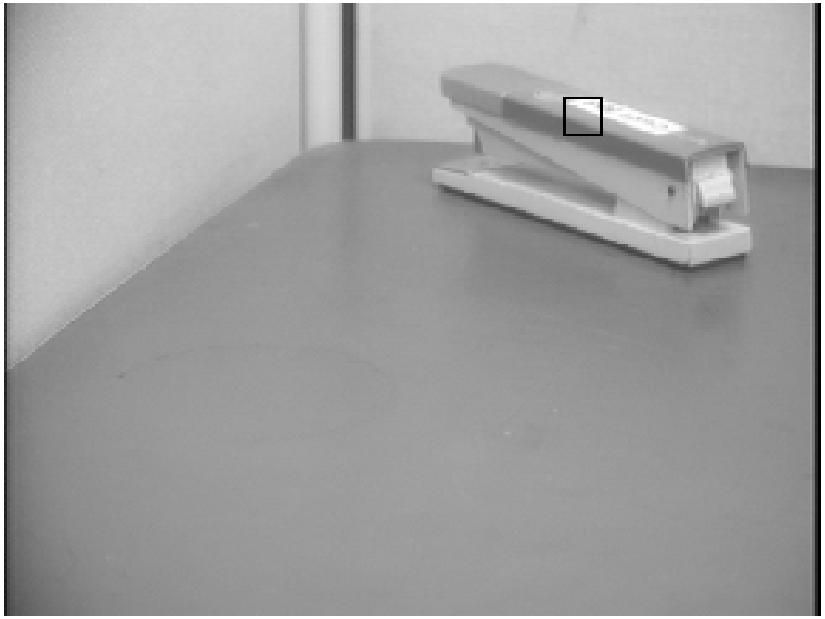
Primary colors of light (additive)
$\Leftrightarrow$ frequencies where the three color receptors are most sensitive
$\Leftrightarrow$ red, green, blue

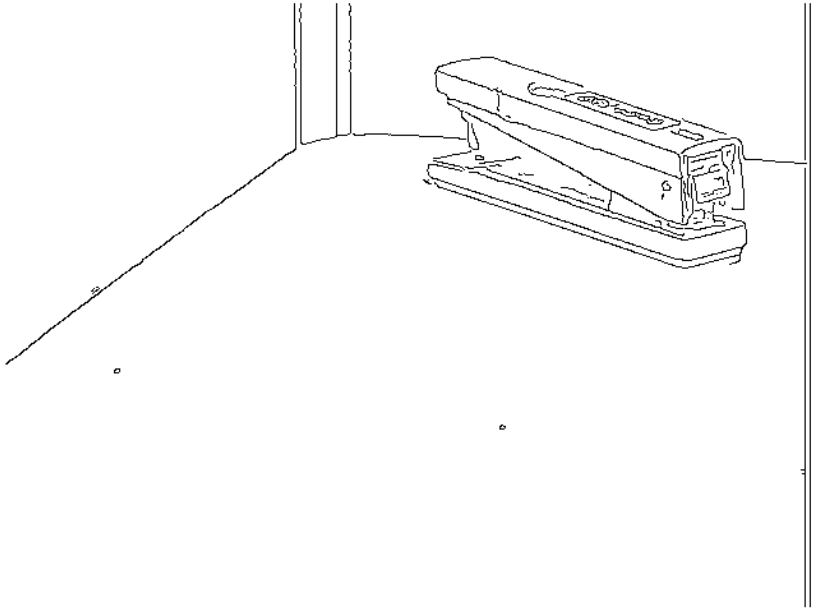


Primary pigments (subtractive)
white minus red = cyan
white minus green = magenta
white minus blue = yellow

"Four color" (or CMYK) printing uses cyan, magenta, yellow, and black.

Edge detection





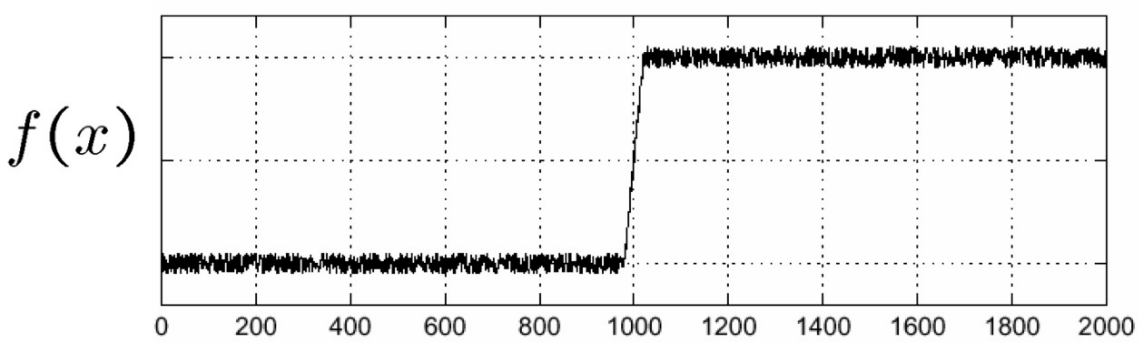
Edges in image $\Leftarrow$ discontinuities in scene:

1) depth
2) surface orientation
3) reflectance (surface markings)
4) illumination (shadows, etc.)

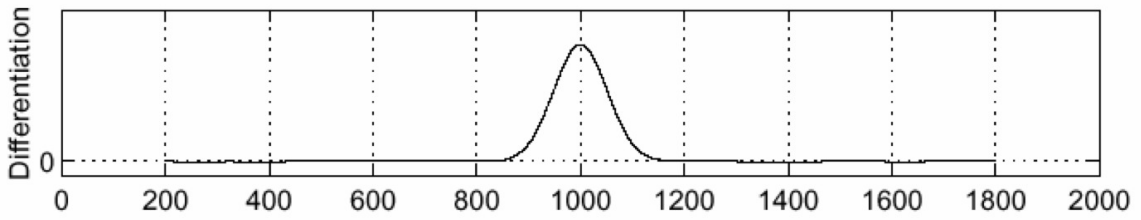
Edges correspond to abrupt changes in brightness or color

Edge detection

Edges correspond to abrupt changes in brightness or color
E.g., a single row or column of an image:



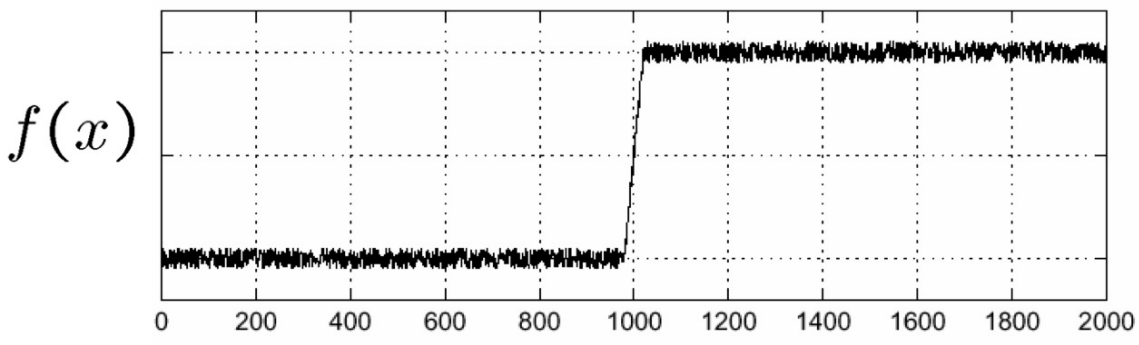
Ideally, we could detect abrupt changes by computing a derivative:



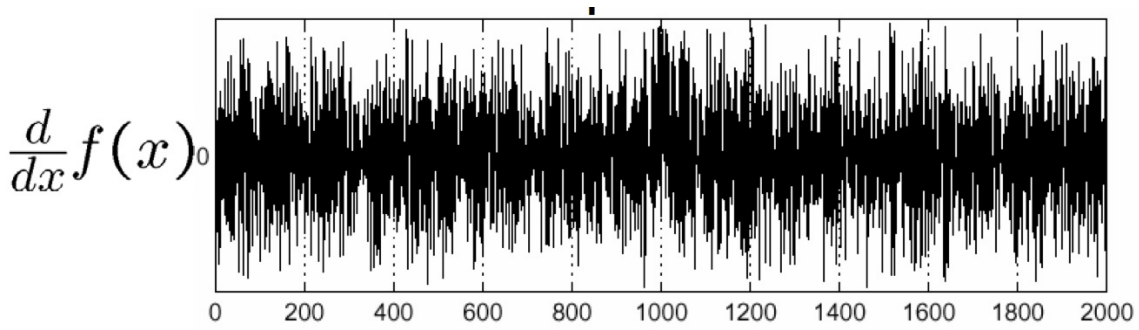
In practice, this has some problems . . .

Noise

Problem: the image is noisy:



The derivative looks like this:



First need to smooth out the noise

Smoothing out the noise

Convolution $g = h \star f$: for each point $f[i, j]$, compute a weighted average $g[i, j]$ of the points near $f[i, j]$

$$g[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u, v] f[i - u, j - v]$$

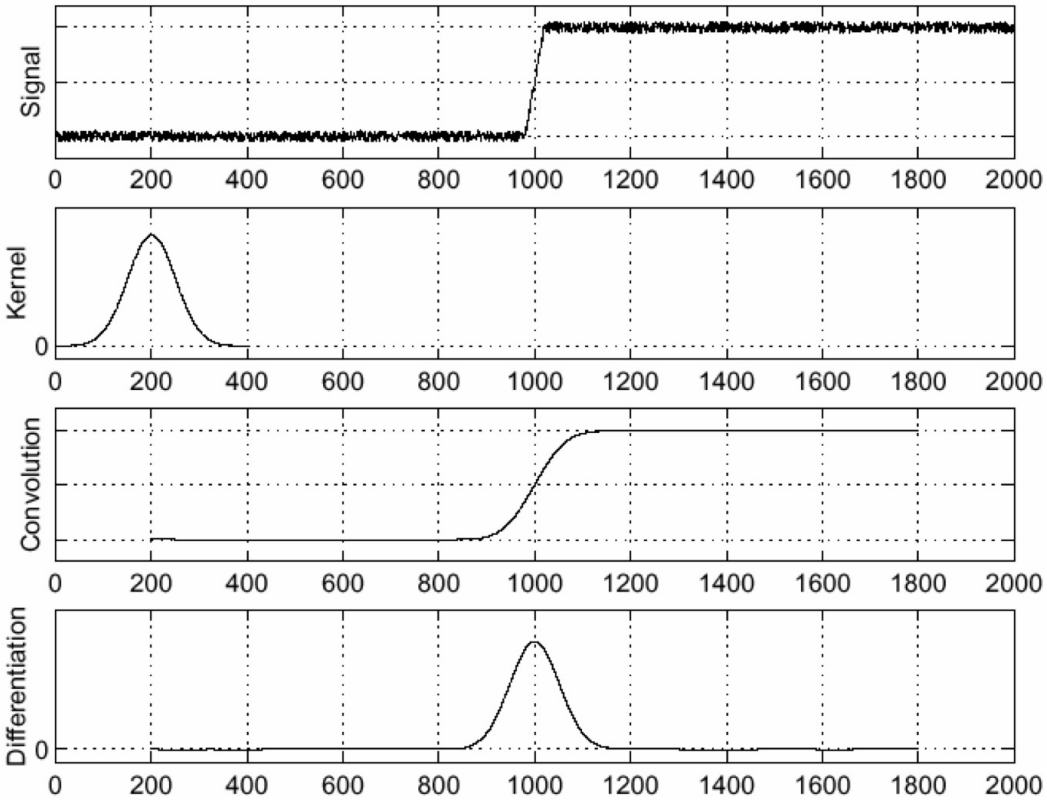
Use convolution to smooth the image, then differentiate:

f

h

$h \star f$

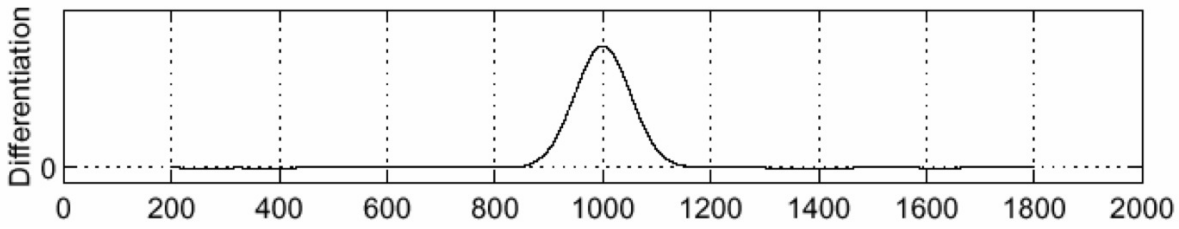
$\frac{\partial}{\partial x}(h \star f)$



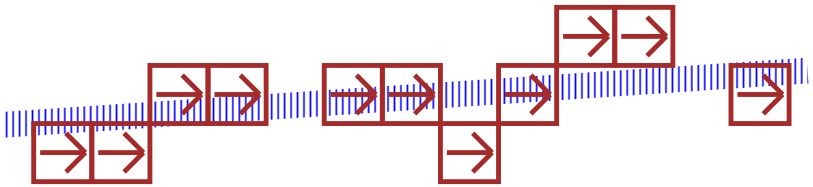
Edge detection

Find all pixels at which the derivative is above some threshold:

$$\frac{\partial}{\partial x}(h \star f)$$



Label above-threshold pixels with edge orientation



Infer "clean" line segments by combining edge pixels with same orientation

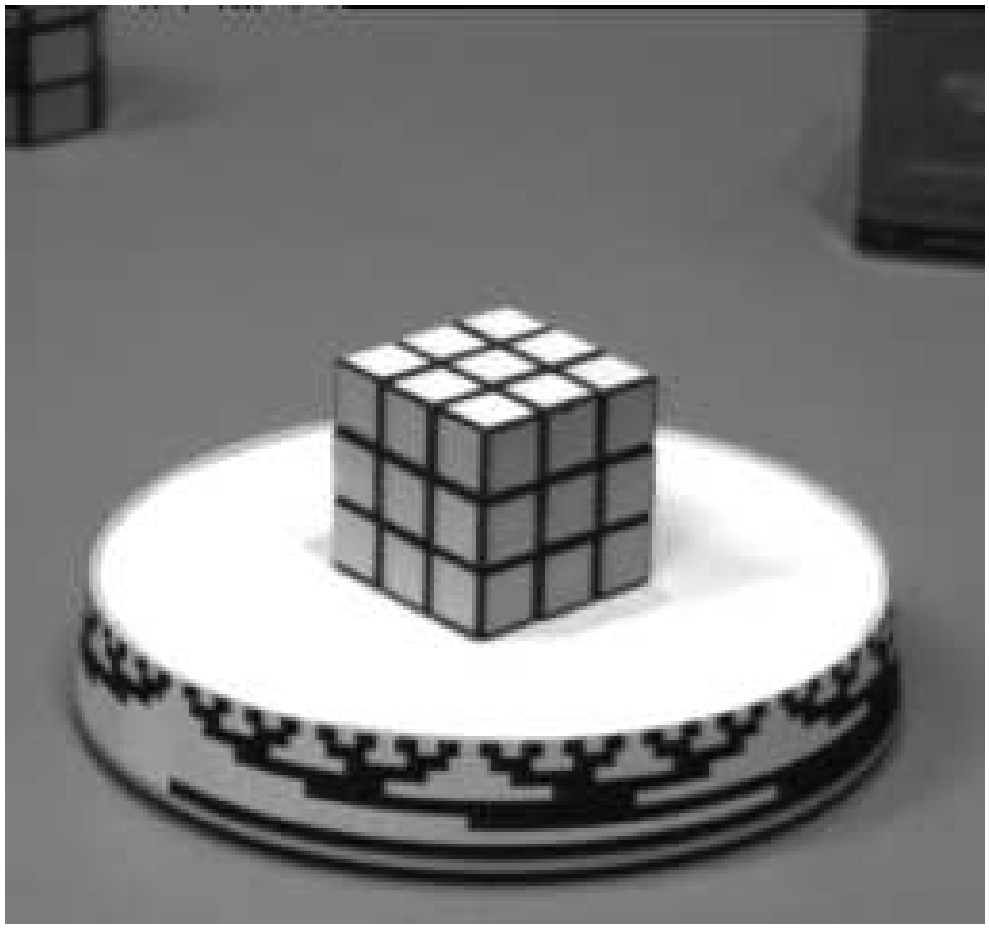
Inferring shape

Can use cues from prior knowledge to help infer shape:

Shape from...	**Assumes**
motion	rigid bodies, continuous motion
stereo	solid, contiguous, non-repeating bodies
texture	uniform texture
shading	uniform reflectance
contour	minimum curvature

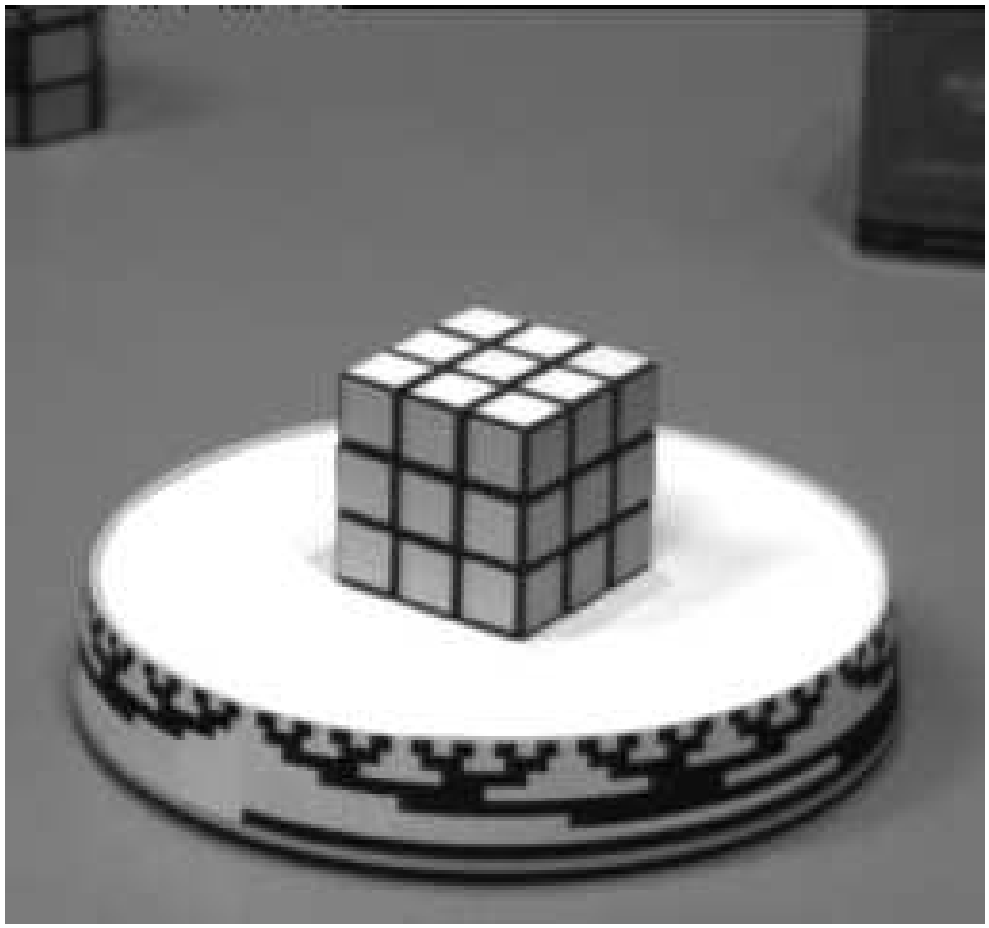
Inferring shape from motion

Is this a 3-d shape? If so, what shape is it?



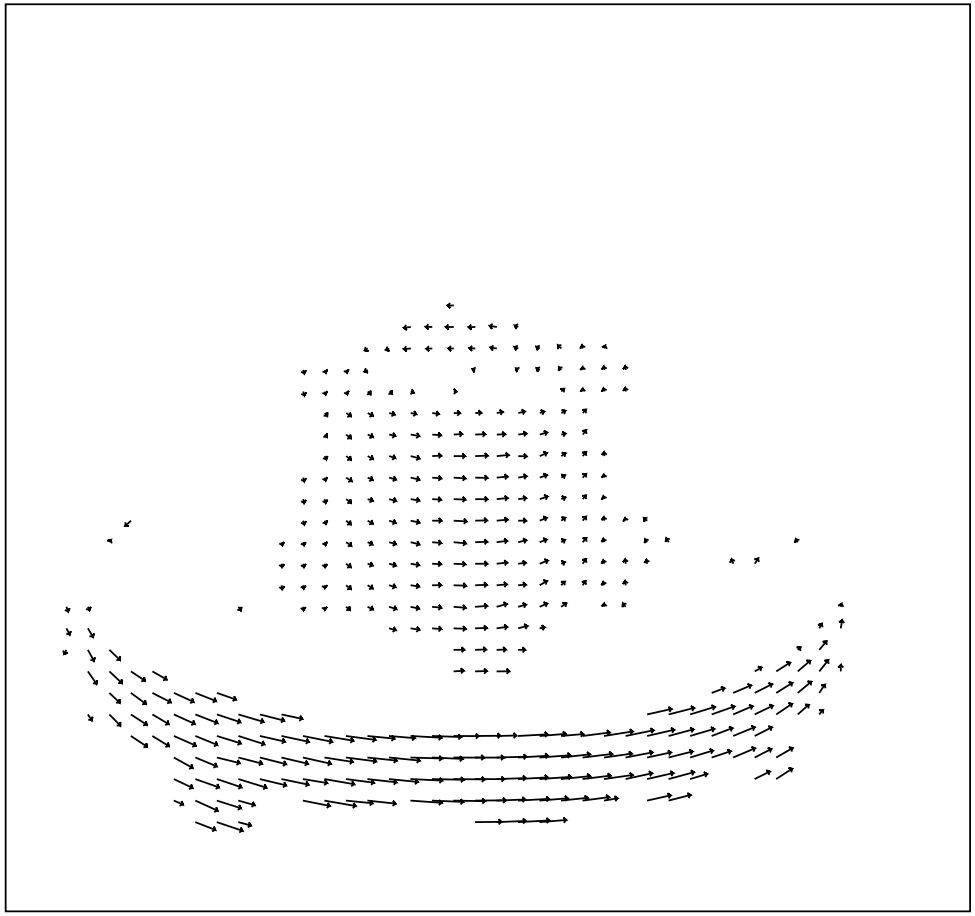
Inferring shape from motion

Rotate the shape slightly



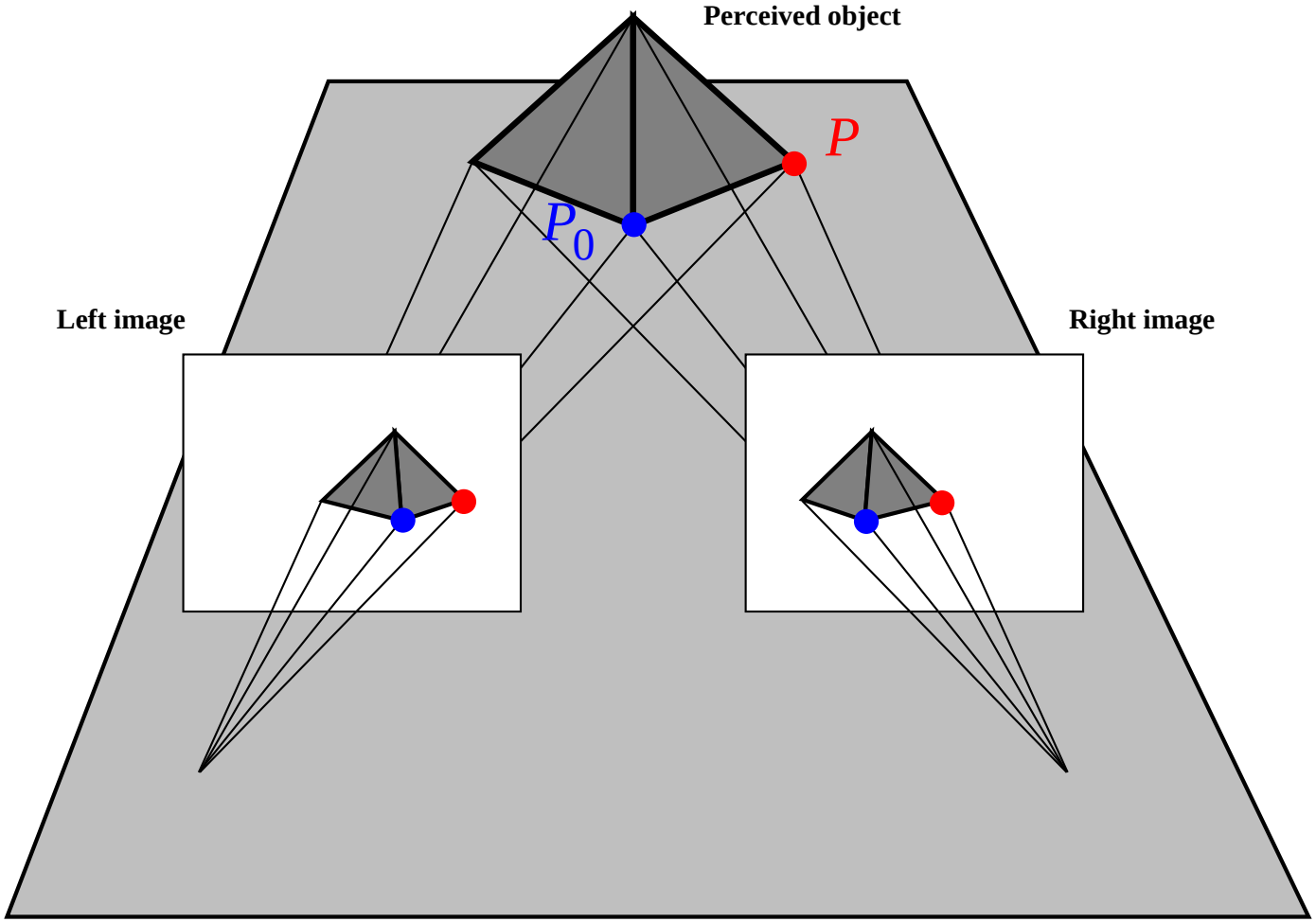
Inferring shape from motion

Match the corresponding pixels in the two images
Use this to infer 3-dimensional locations of the pixels

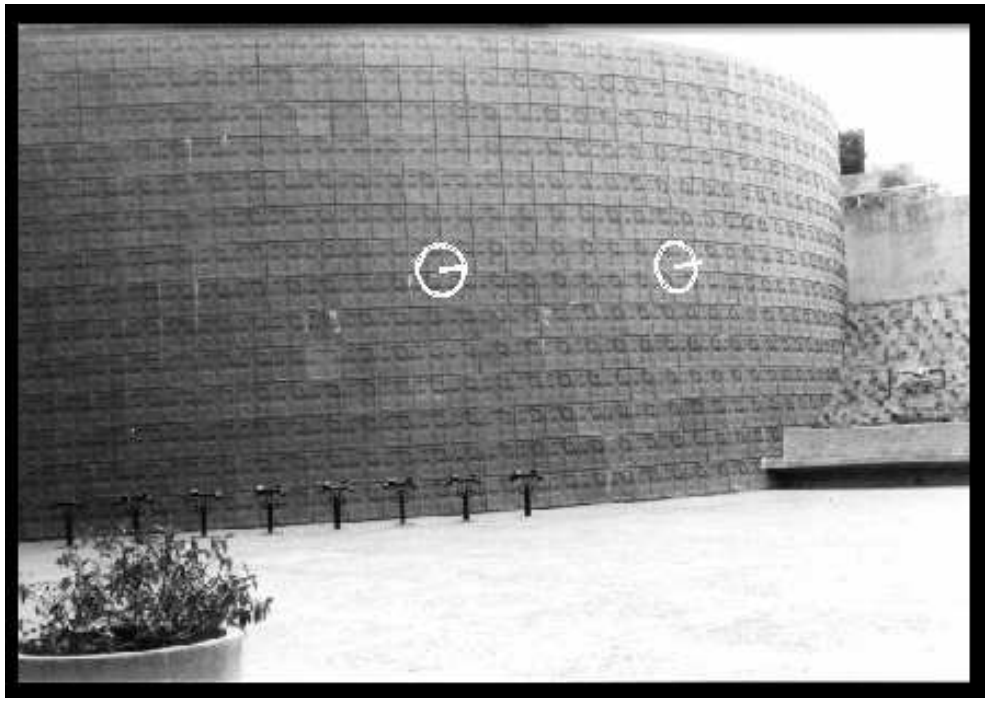


Inferring shape using stereo vision

Can do something similar using stereo vision



Inferring shape from texture



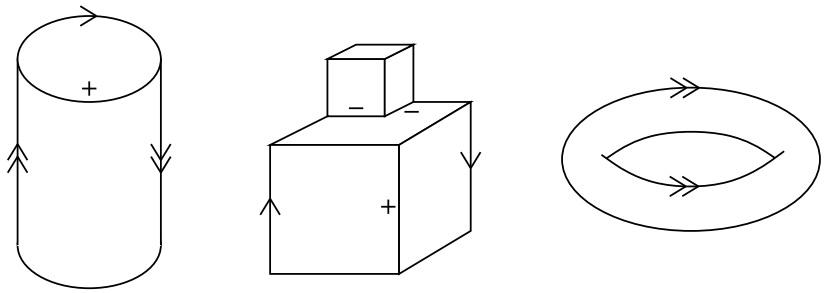
Idea: assume actual texture is uniform, compute surface shape that would produce this distortion

Similar idea works for shading—assume uniform reflectance, etc.—**but** interreflections give nonlocal computation of perceived intensity

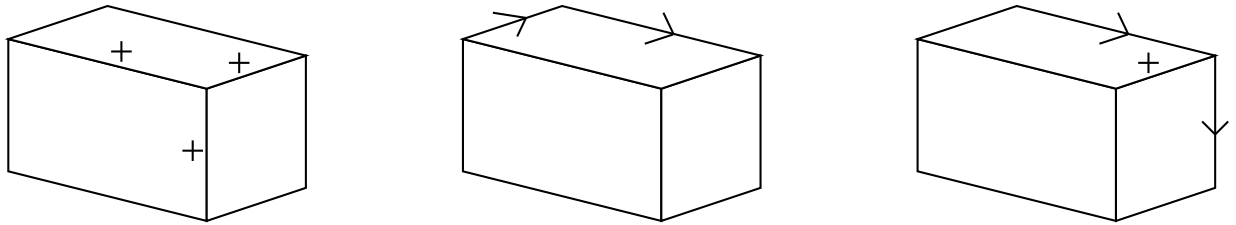
$\Rightarrow$ hollows seem shallower than they really are

Inferring shape from edges

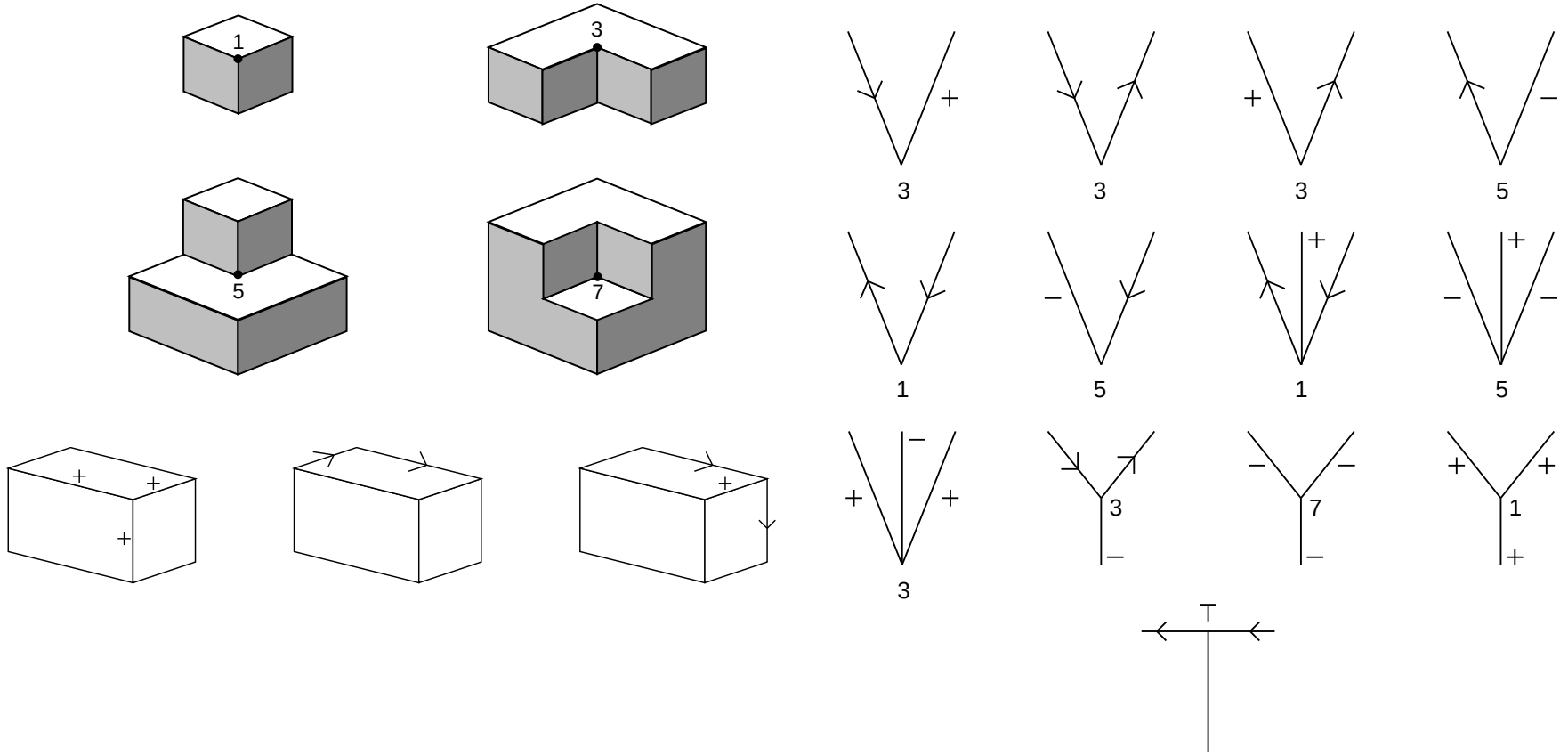
Sometimes we can infer shape from properties of the edges and vertices:



Let's only consider solid polyhedral objects with trihedral vertices
$\Rightarrow$ Only certain types of labels are possible, e.g.,

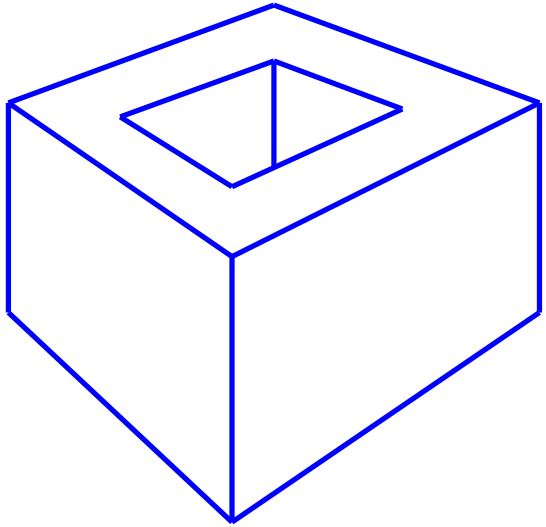


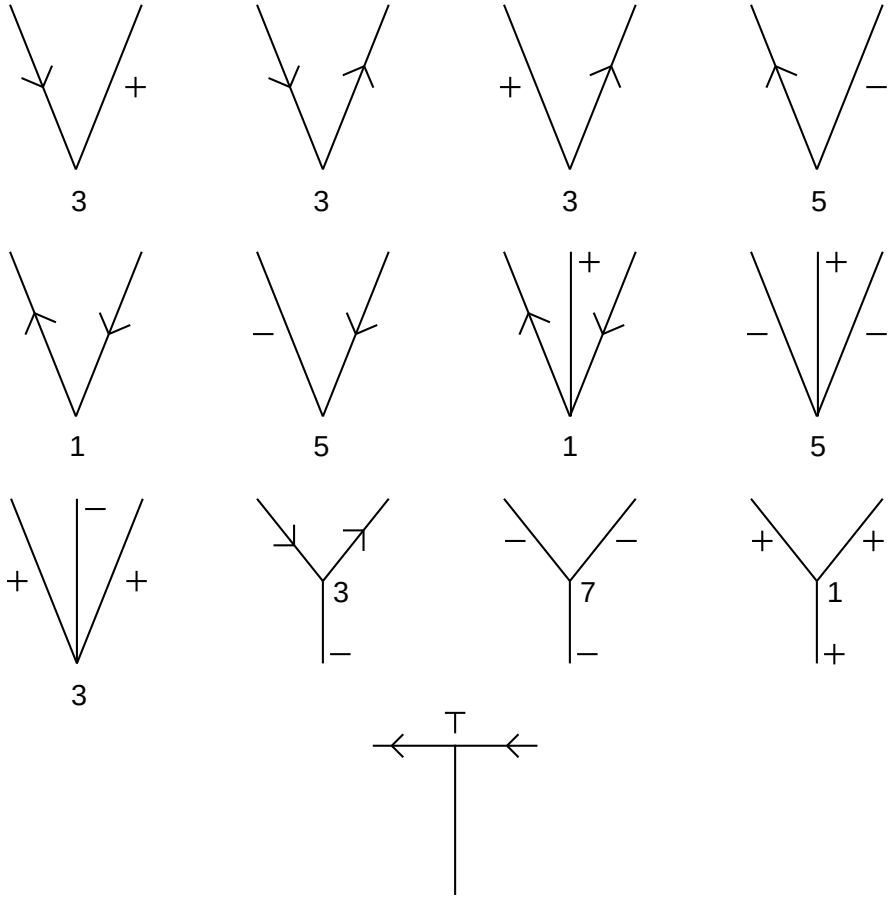
All possible combinations of vertex/edge labels



Given a line drawing, use constraint-satisfaction to assign labels to edges

Vertex/edge labeling example

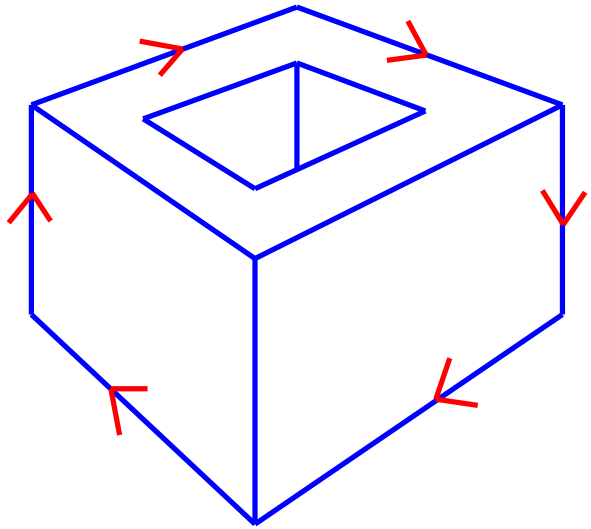


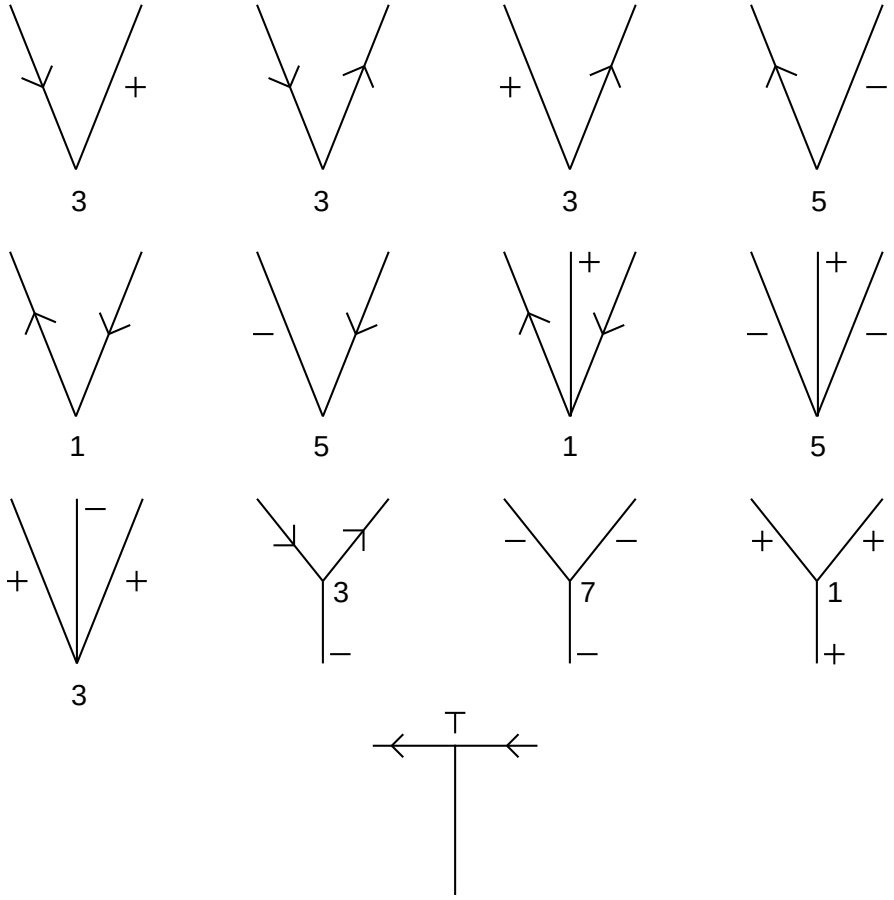


Use a constraint-satisfaction algorithm to assign labels to edges

variables = edges, constraints = possible node configurations

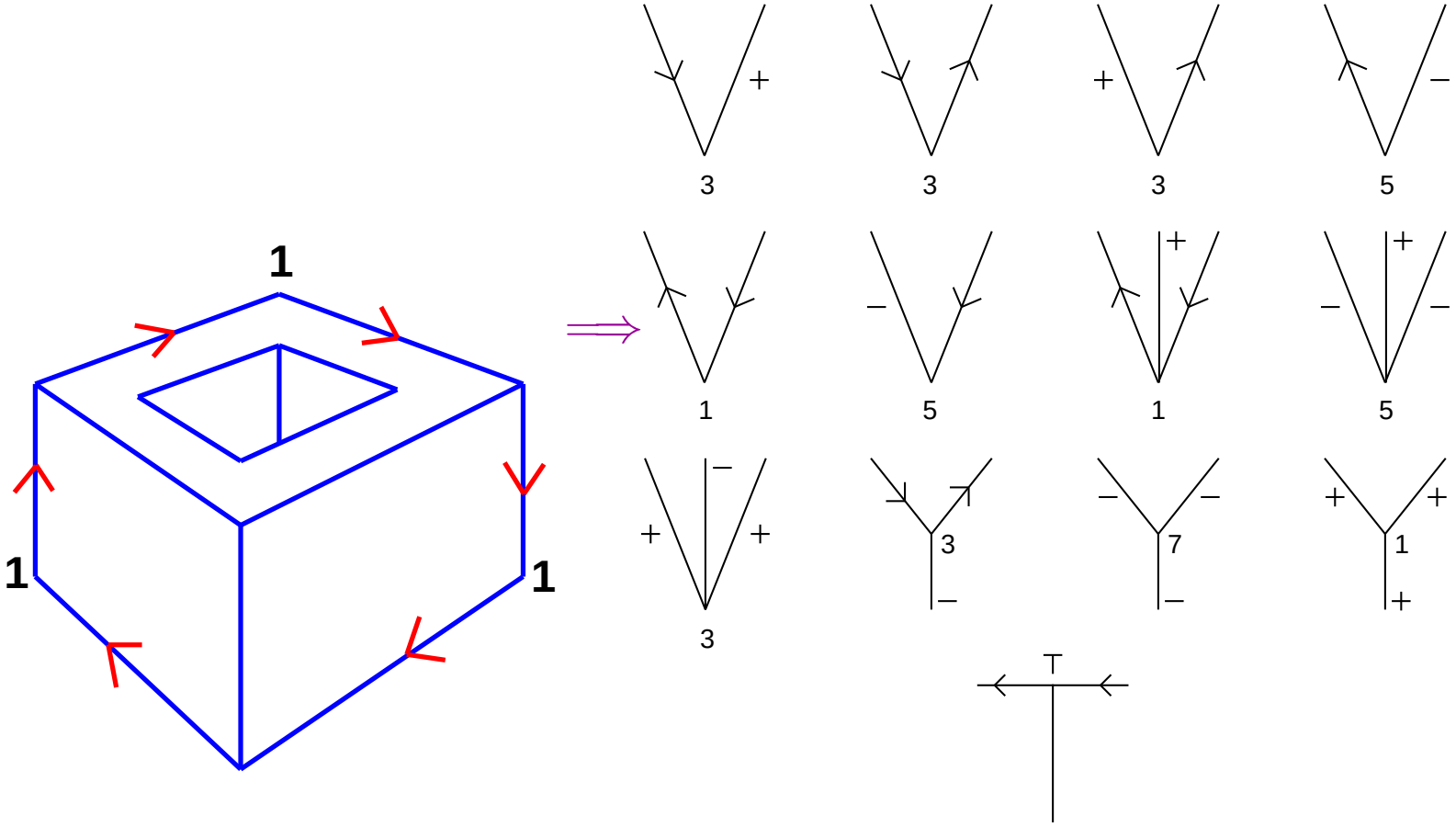
Vertex/edge labeling example



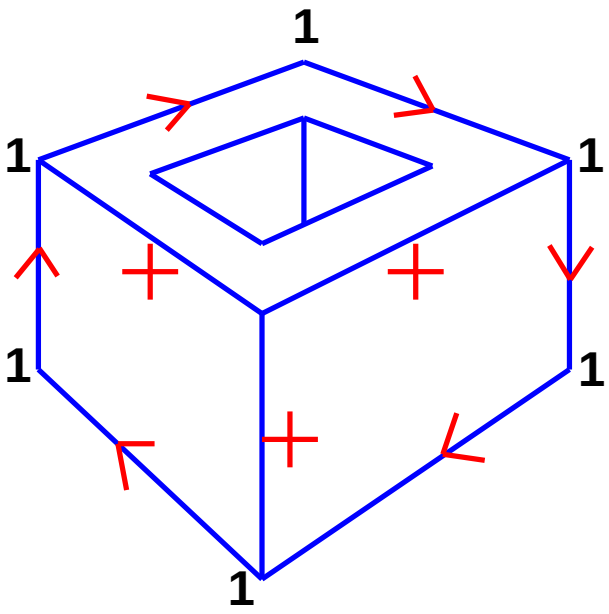


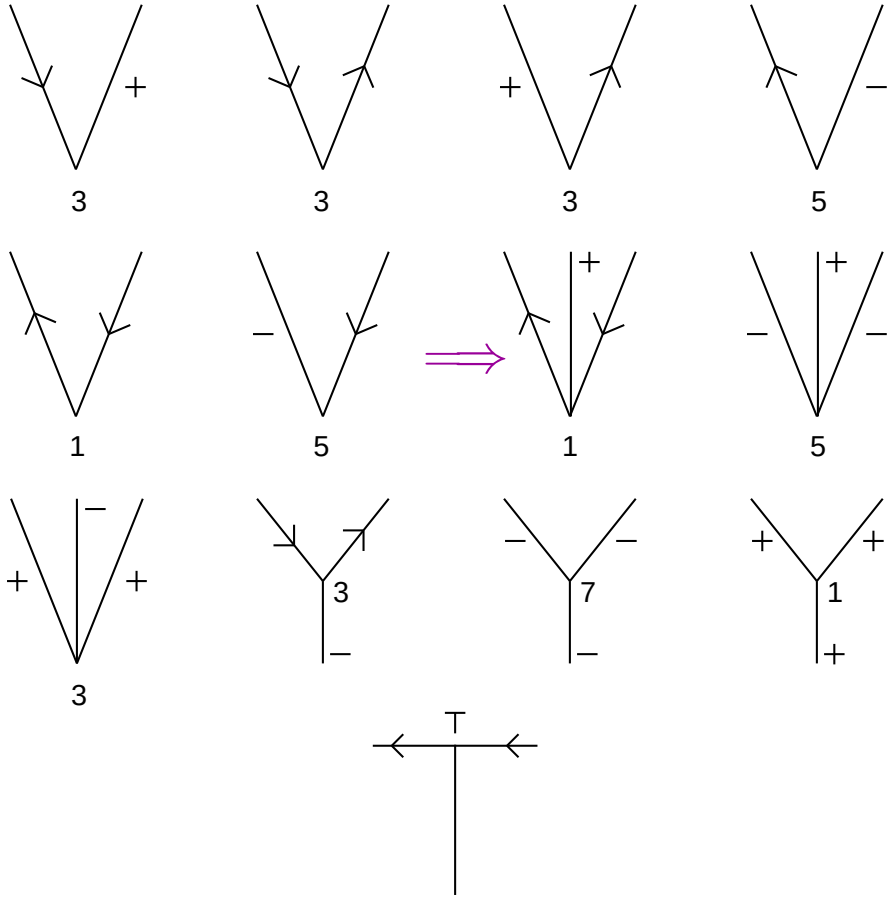
Start by putting clockwise arrows on the boundary

Vertex/edge labeling example

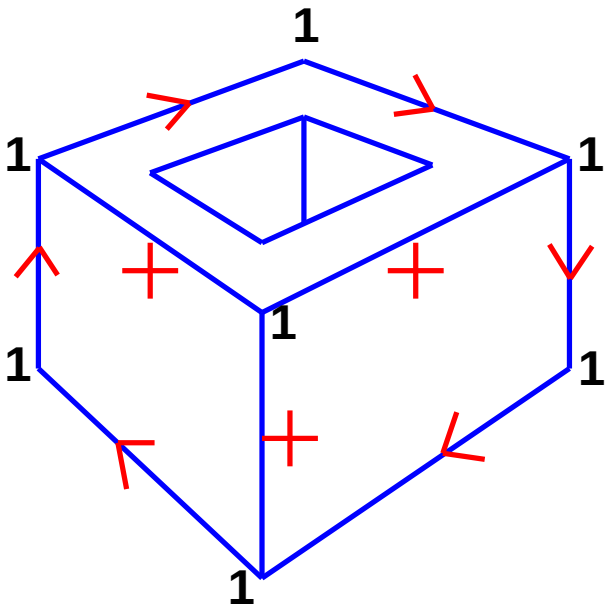


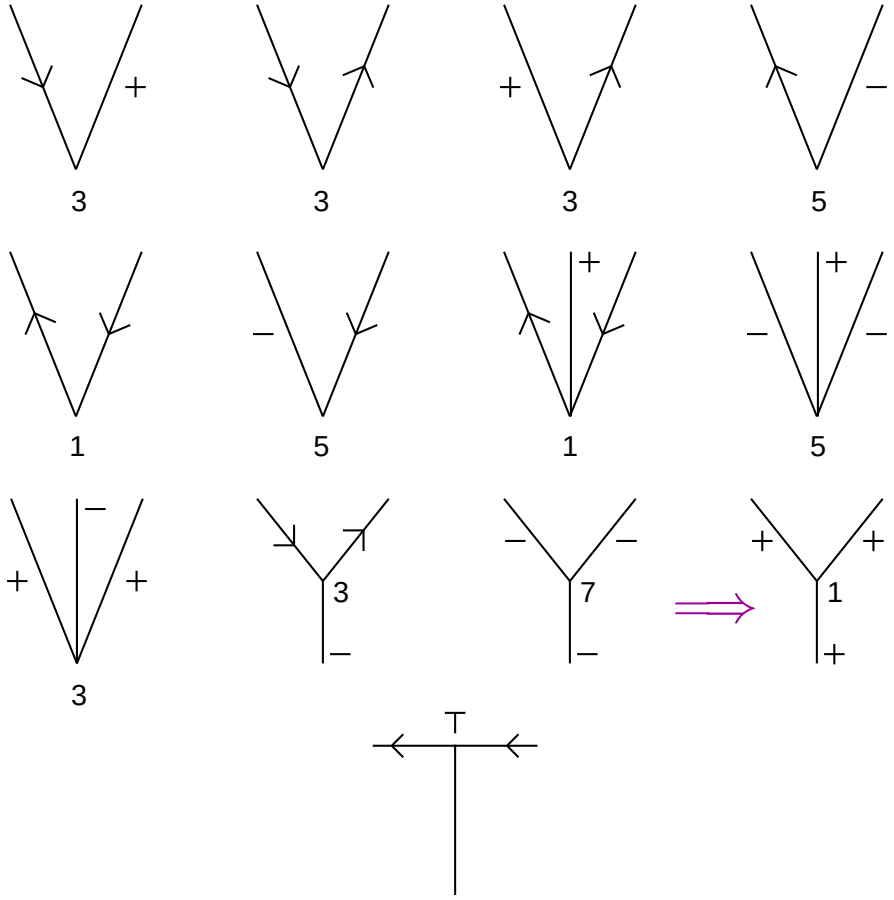
Vertex/edge labeling example



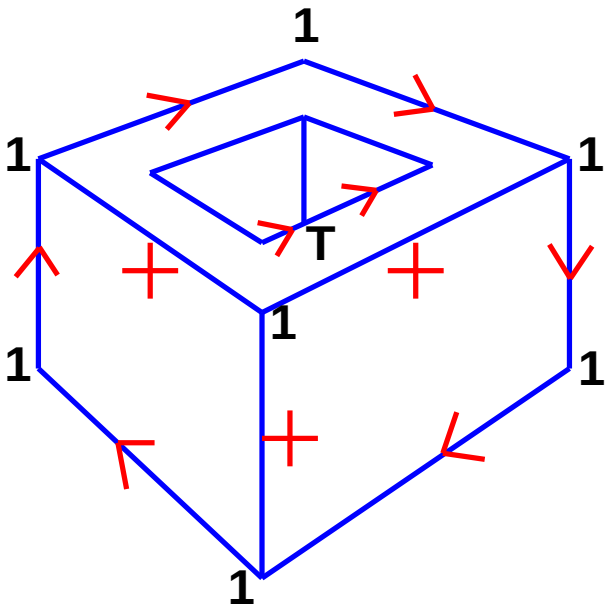


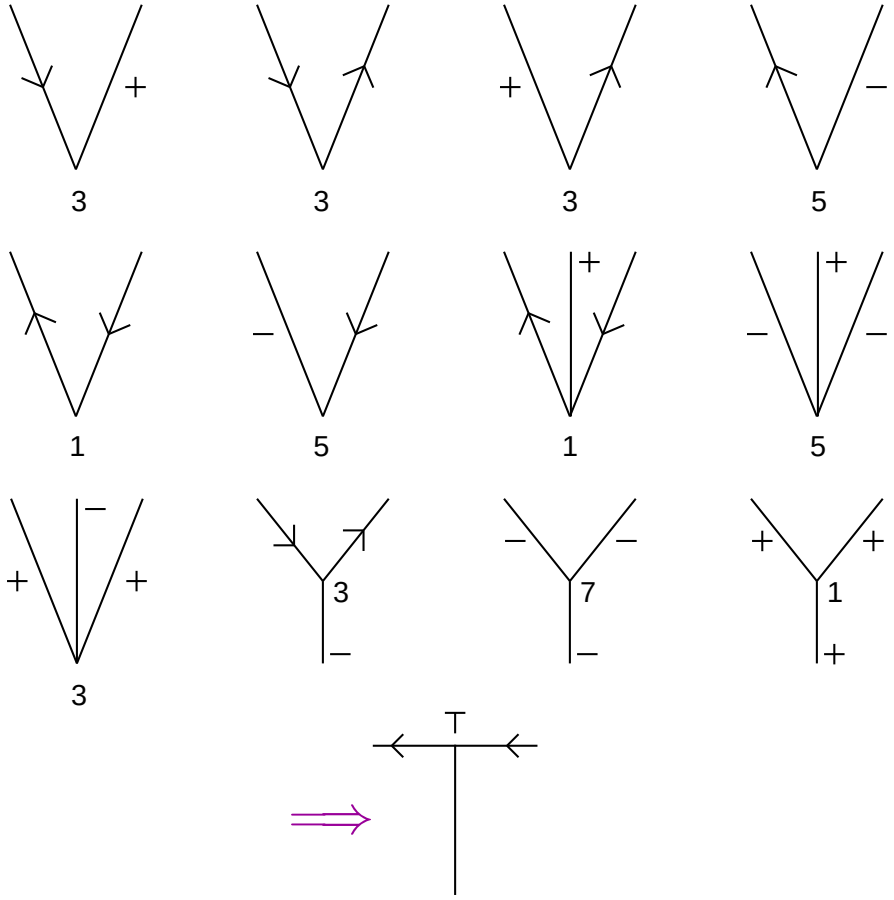
Vertex/edge labeling example



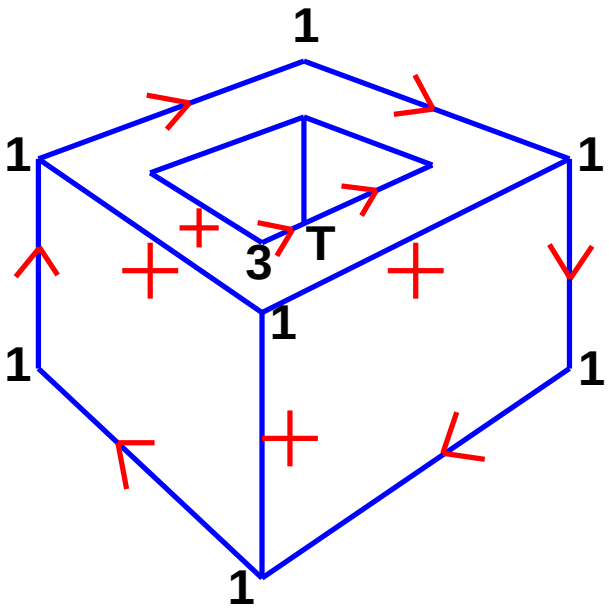


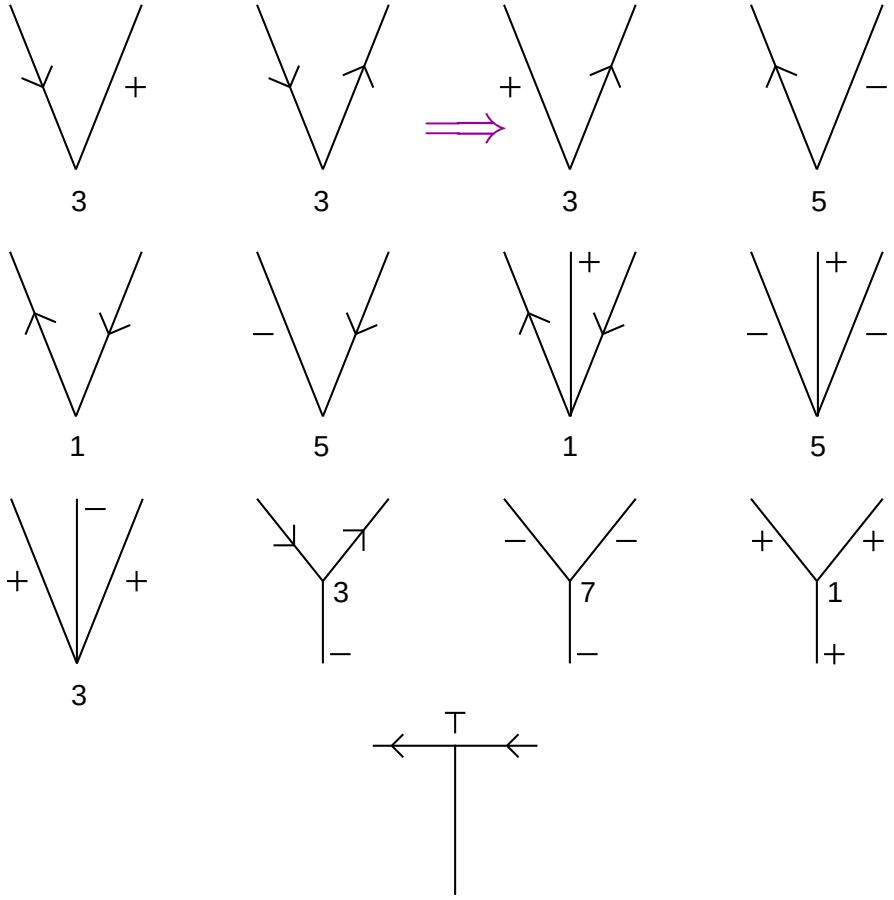
Vertex/edge labeling example



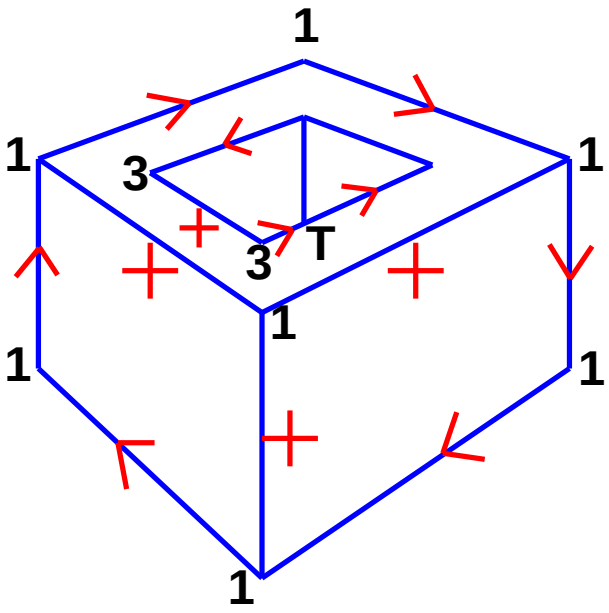


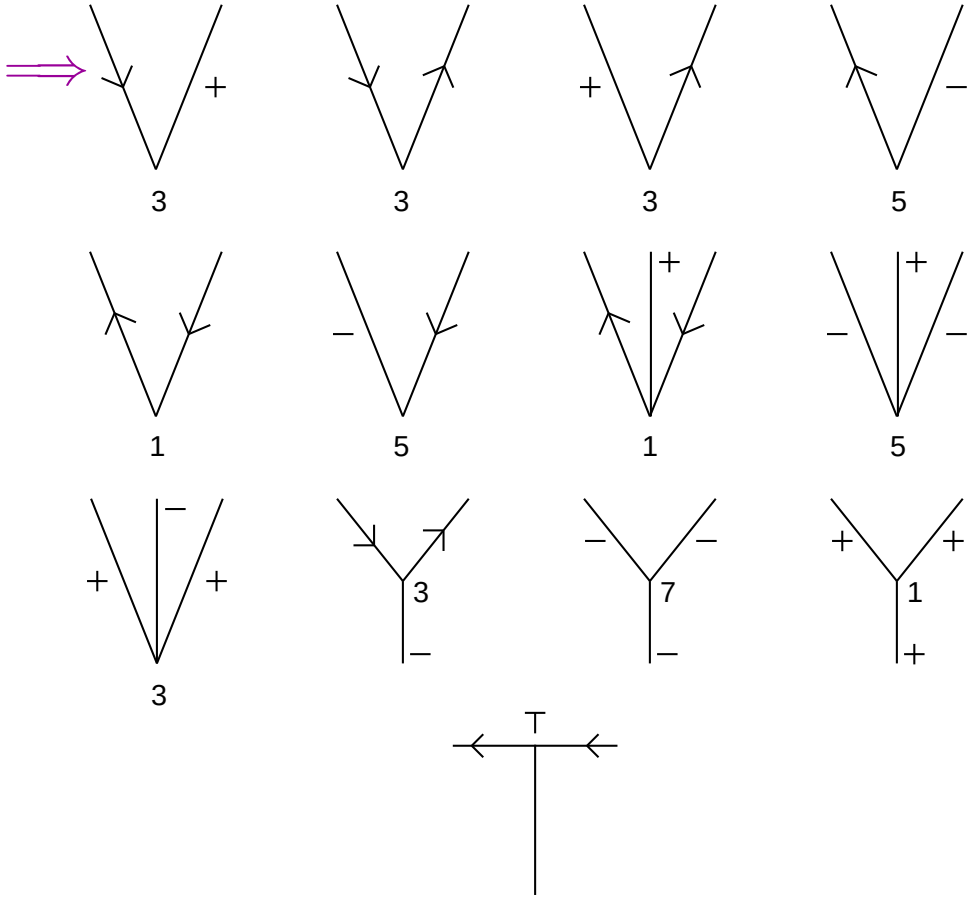
Vertex/edge labeling example



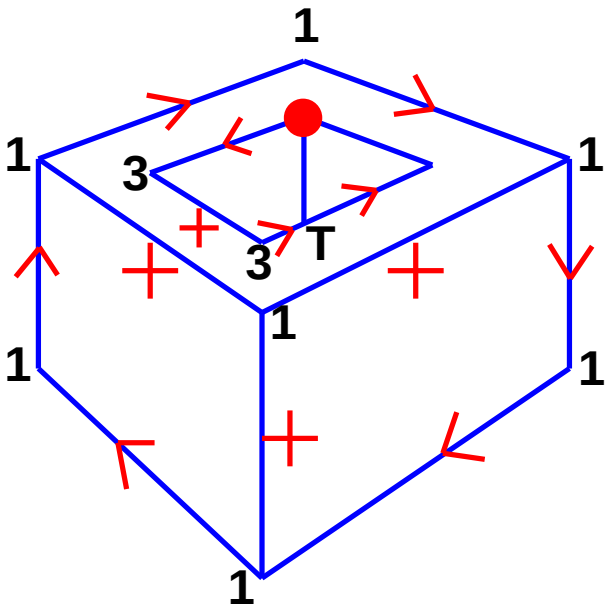


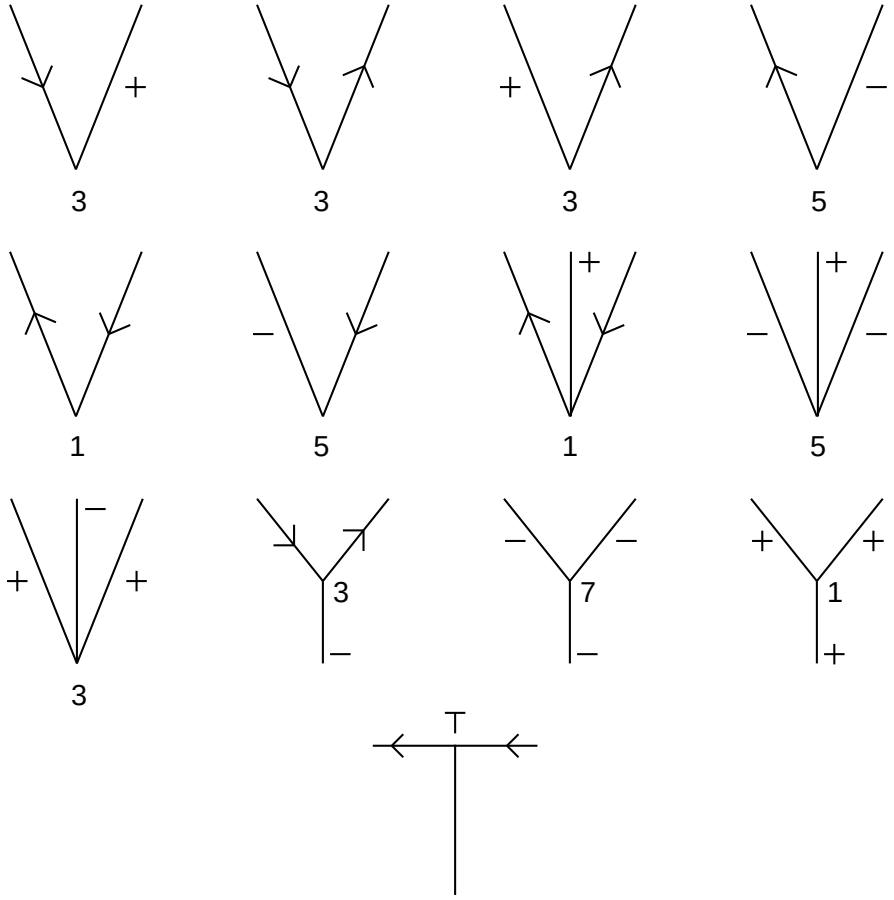
Vertex/edge labeling example





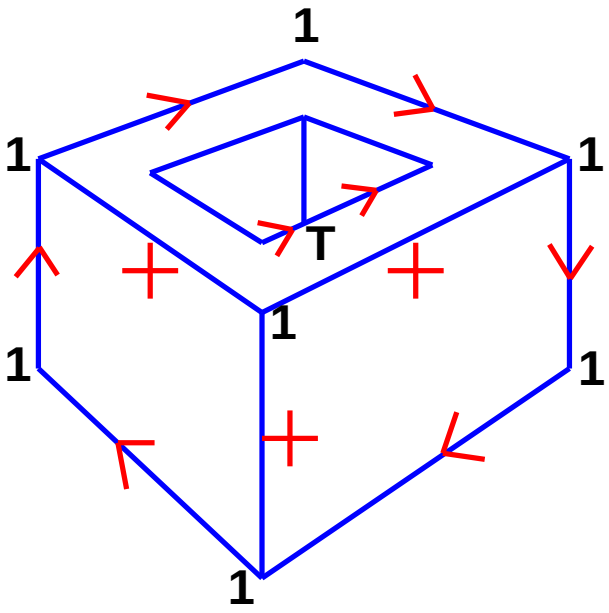
Vertex/edge labeling example

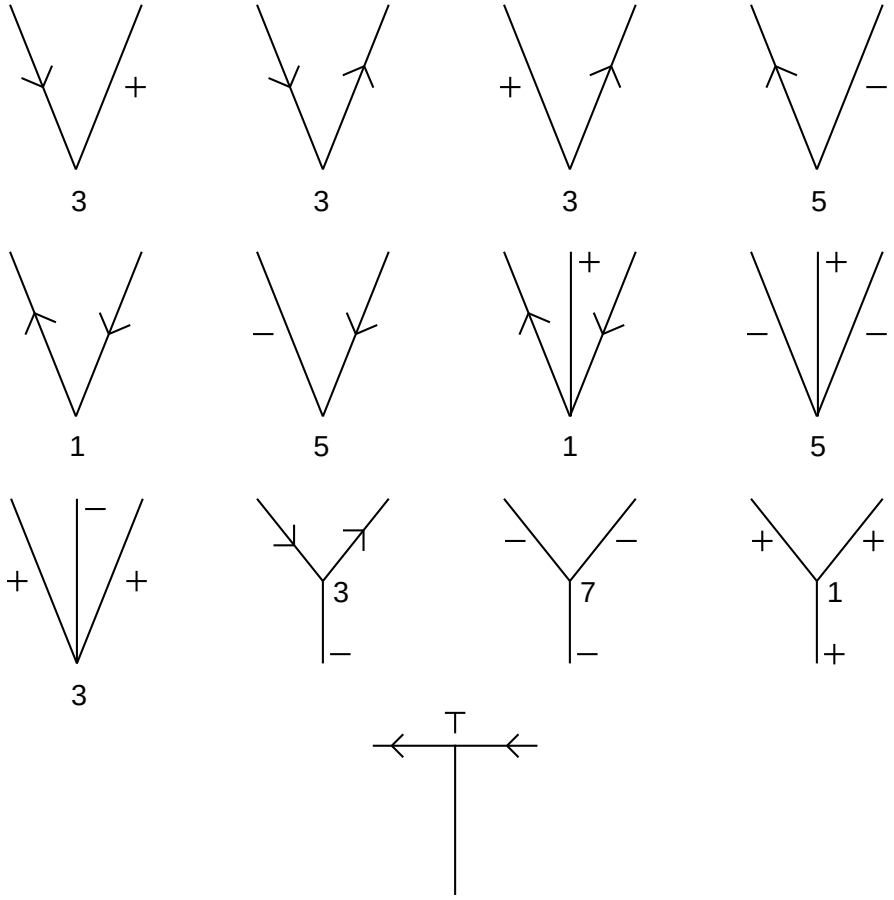




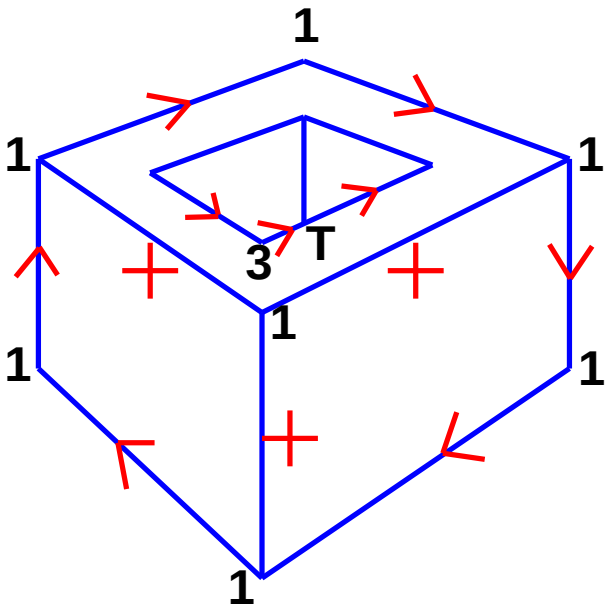
Need to backtrack

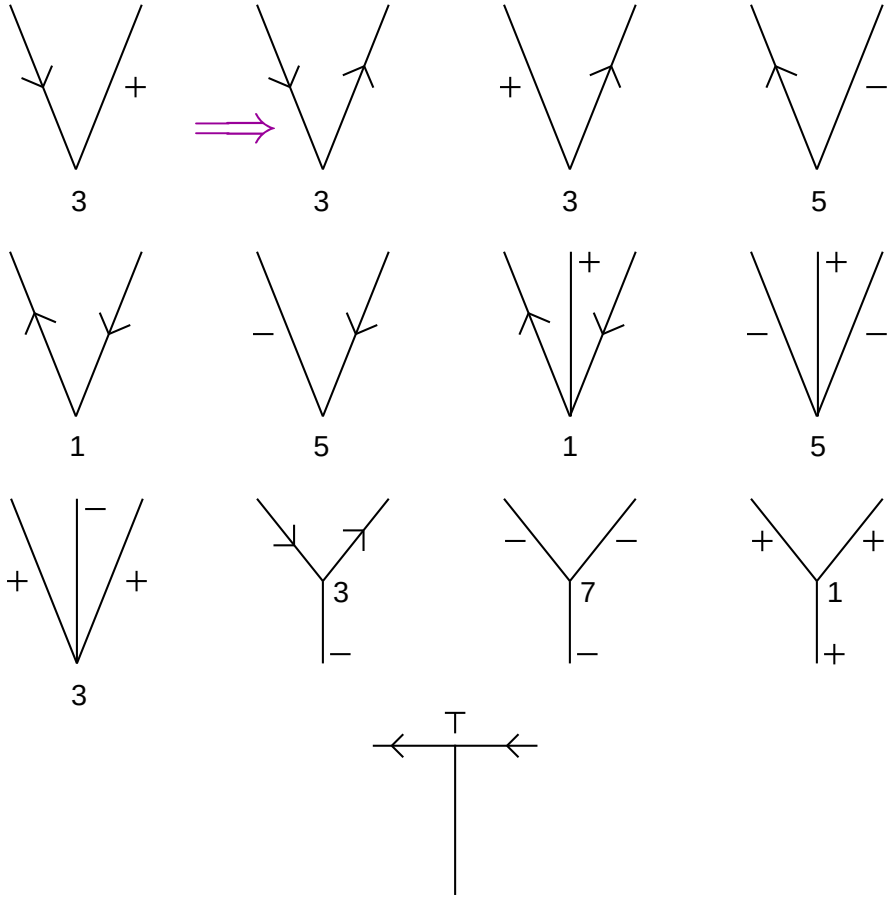
Vertex/edge labeling example



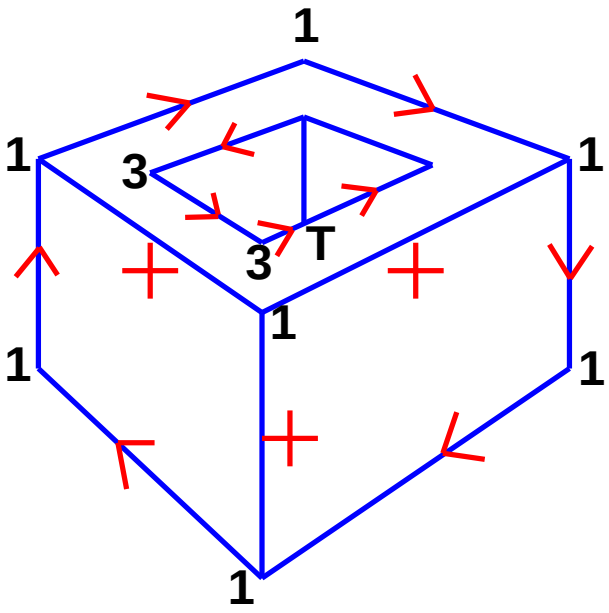


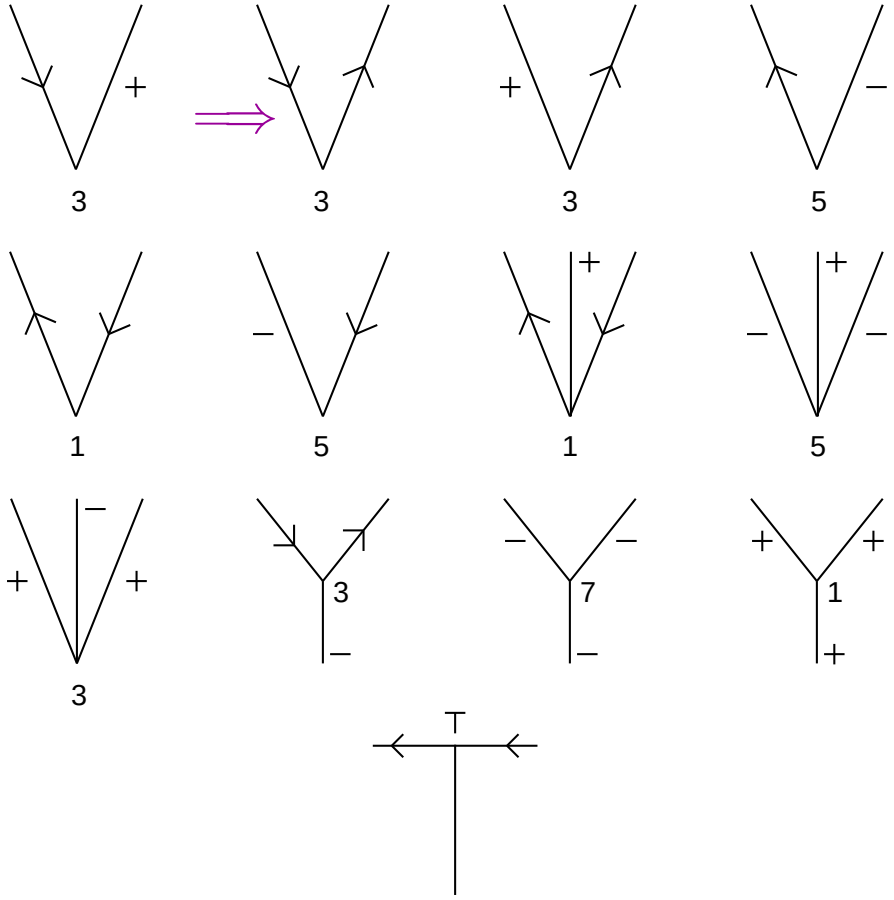
Vertex/edge labeling example



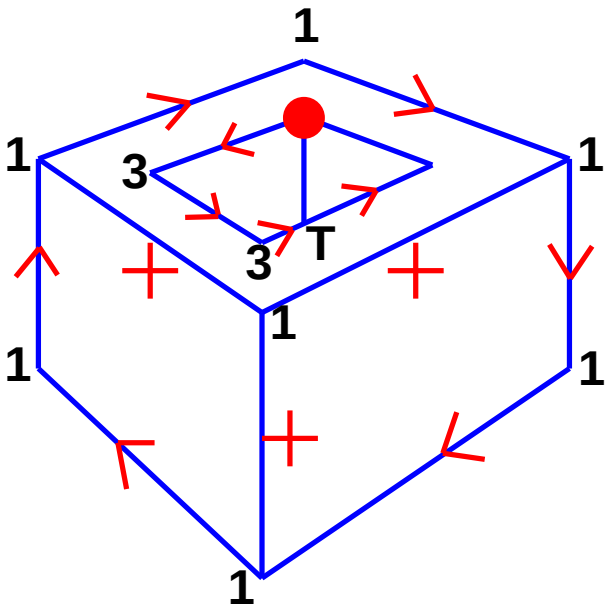


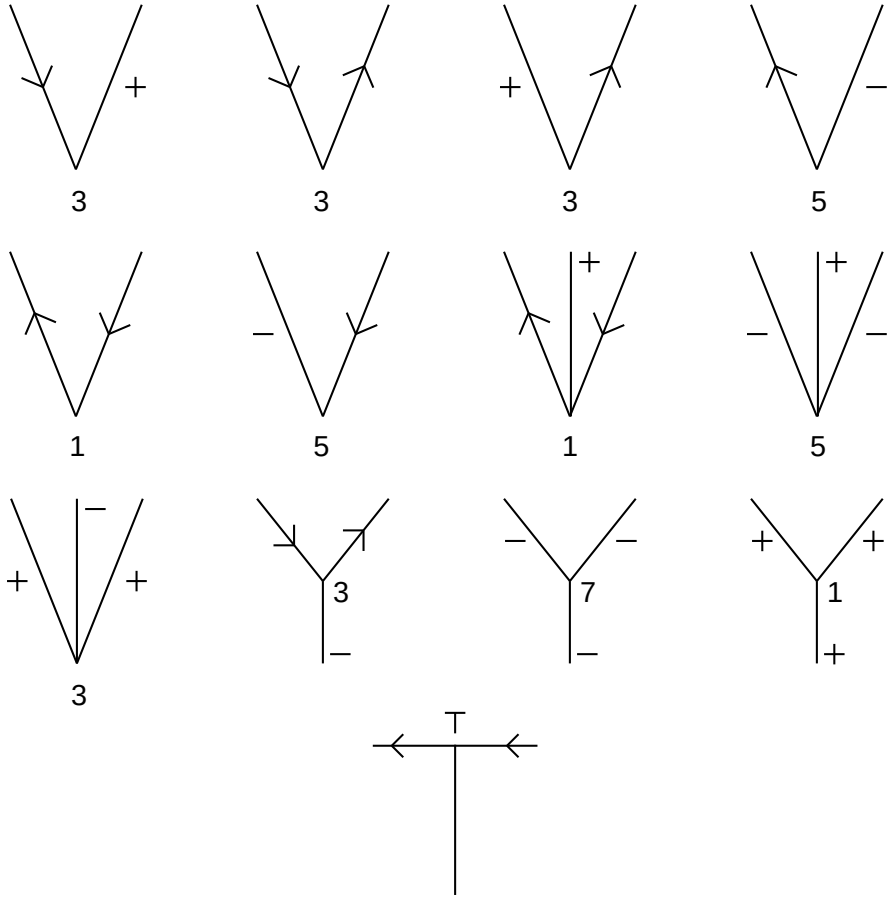
Vertex/edge labeling example





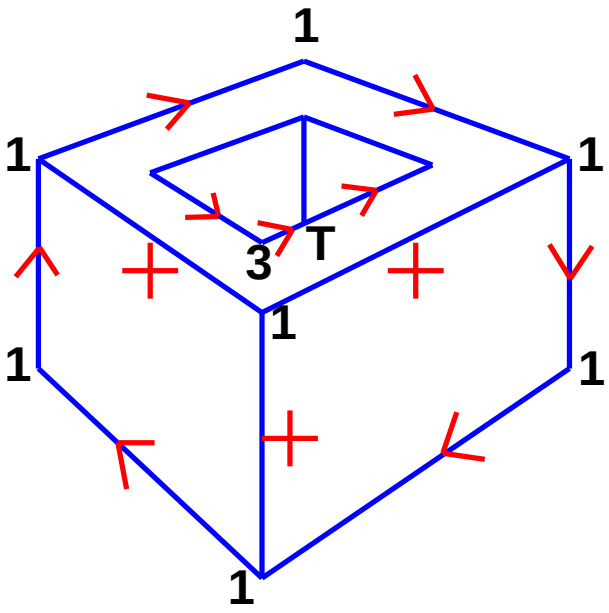
Vertex/edge labeling example

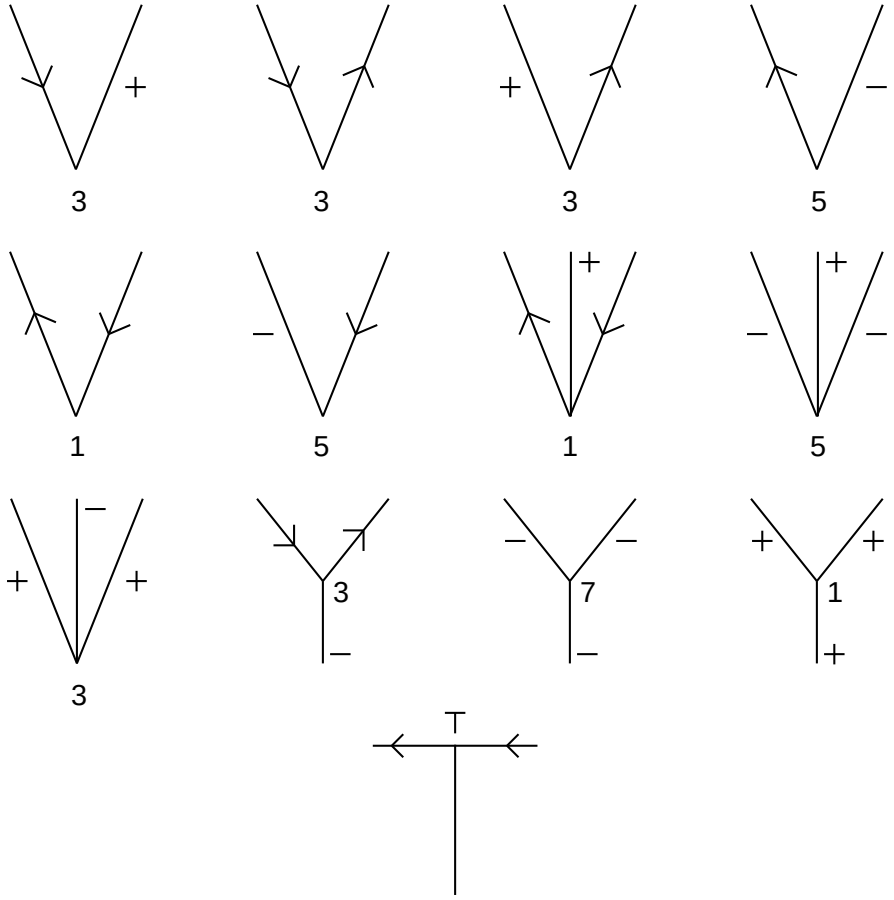




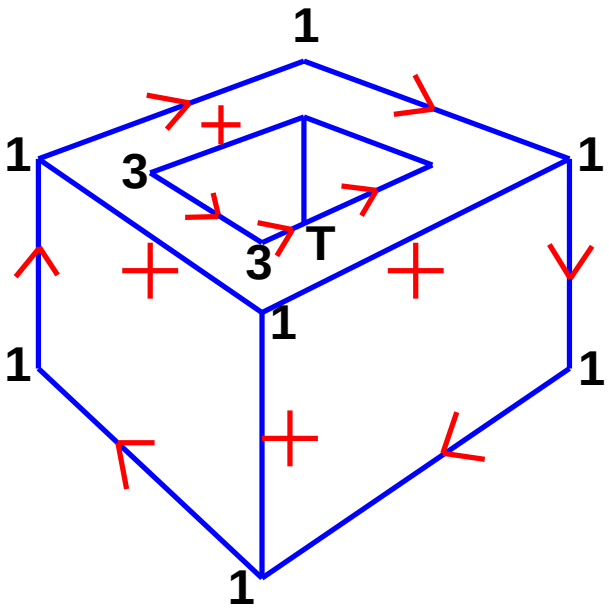
Need to backtrack again

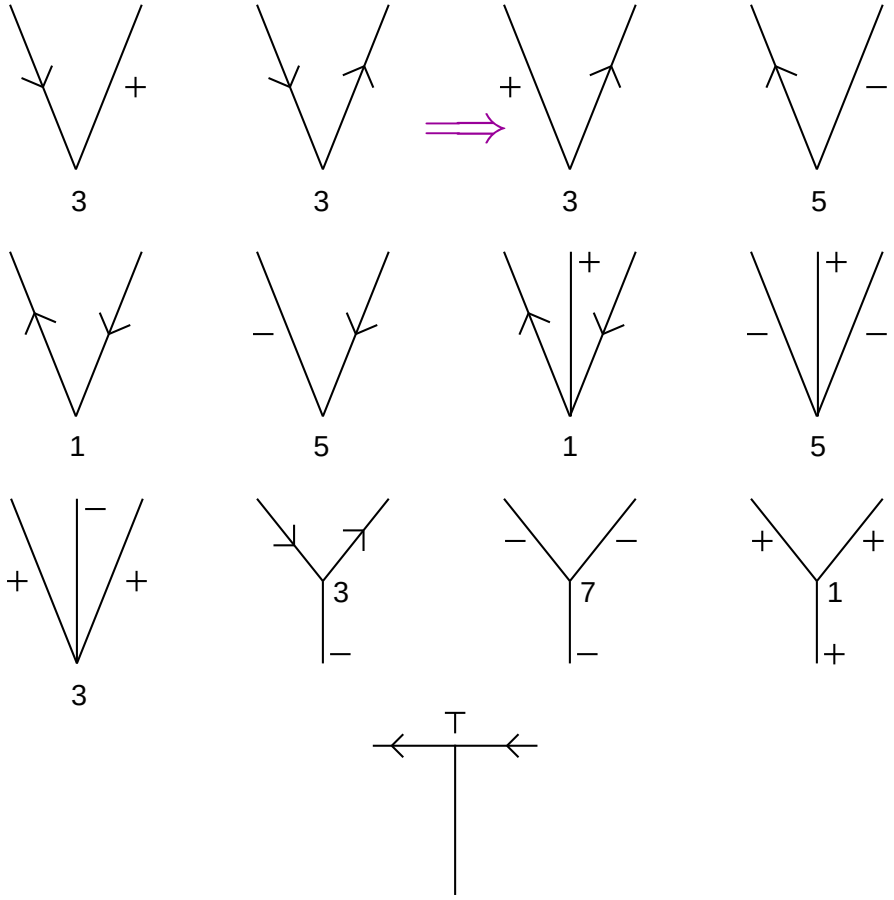
Vertex/edge labeling example



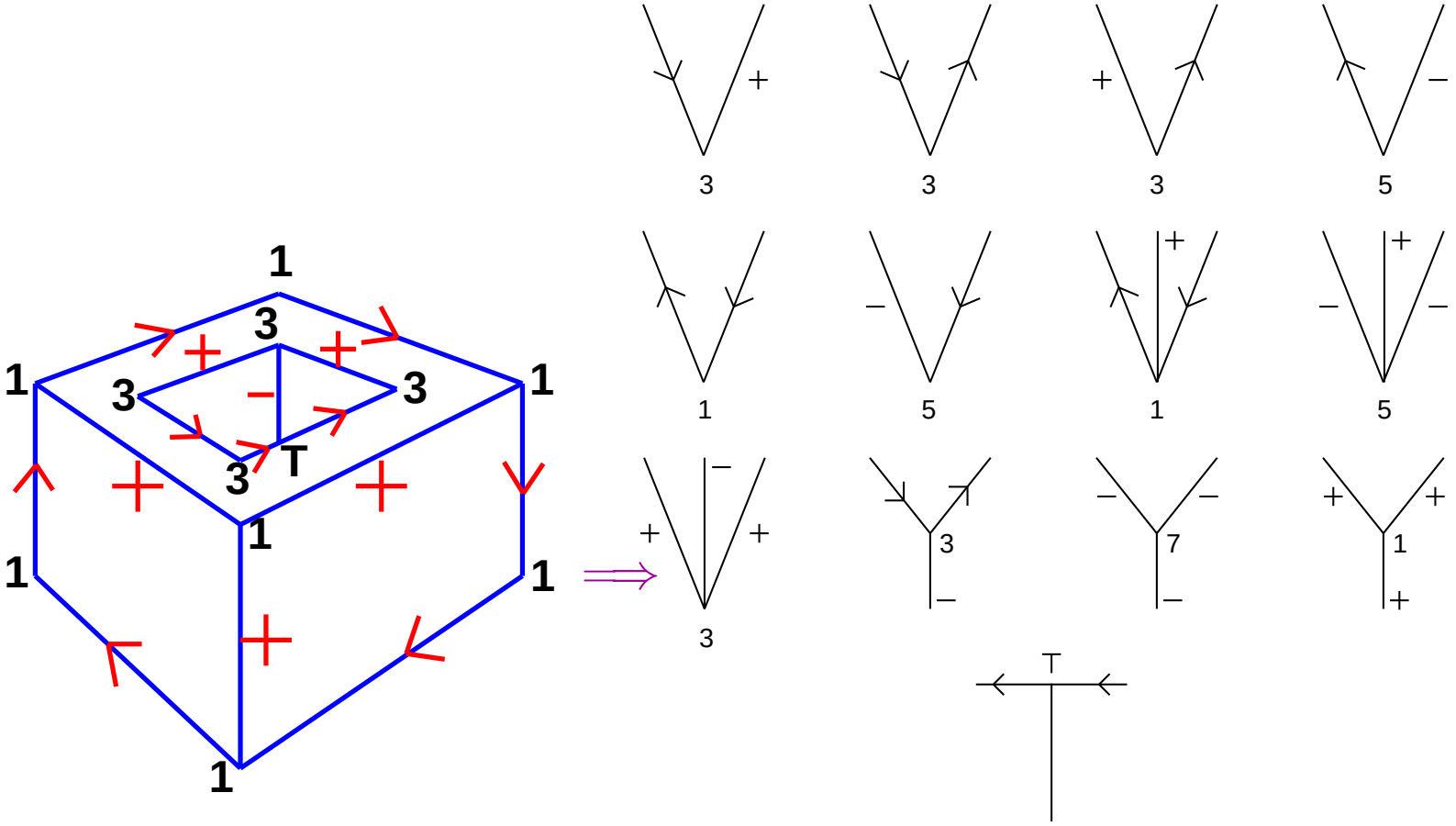


Vertex/edge labeling example





Vertex/edge labeling example



Found a consistent labeling

Object recognition

Simple idea:

- extract 3-D shapes from image
- match against "shape library"

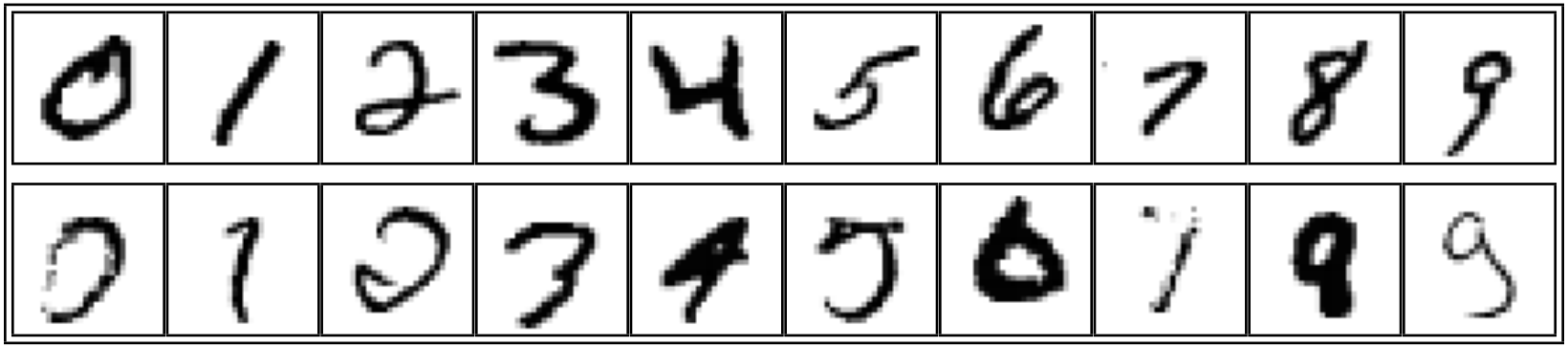
Problems:

- extracting curved surfaces from image
- representing shape of extracted object
- representing shape and variability of library object classes
- improper segmentation, occlusion
- unknown illumination, shadows, markings, noise, complexity, etc.

Approaches:

- index into library by measuring invariant properties of objects
- alignment of image feature with projected library object feature
- match image against multiple stored views (**aspects**) of library object
- machine learning methods based on image statistics

Handwritten digit recognition



3-nearest-neighbor = 2.4% error
400–300–10 unit MLP (multi-layer perceptron) = 1.6% error
LeNet: 768–192–30–10 unit MLP = 0.9% error

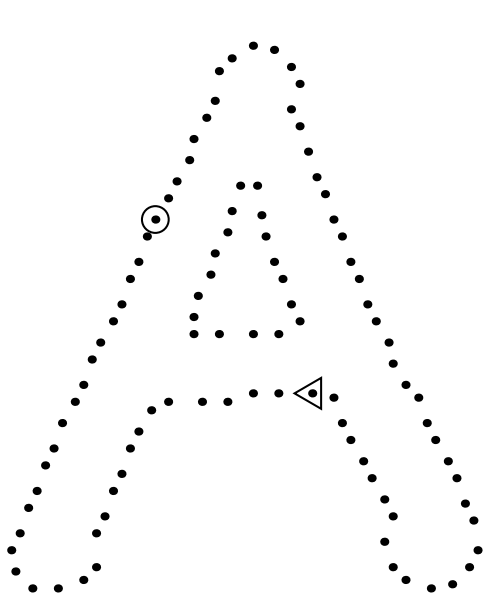
Current best (kernel machines, vision algorithms) $\approx$ 0.6% error

- ♢ A kernel machine is a statistical learning algorithm
- ♢ The "kernel" is a particular kind of vector function (see Section 20.6)

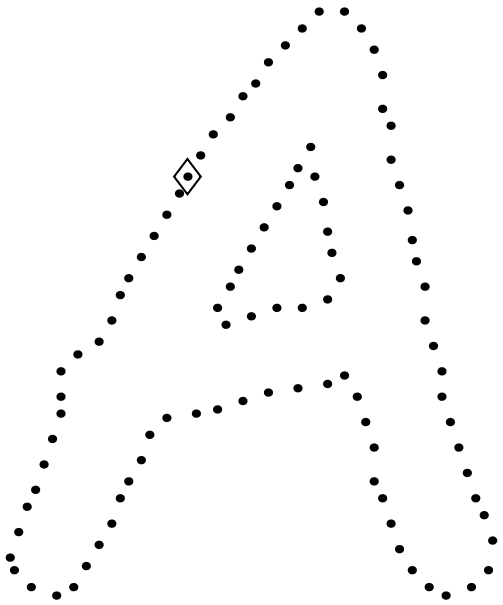
Shape-context matching

Do two shapes represent the same object?

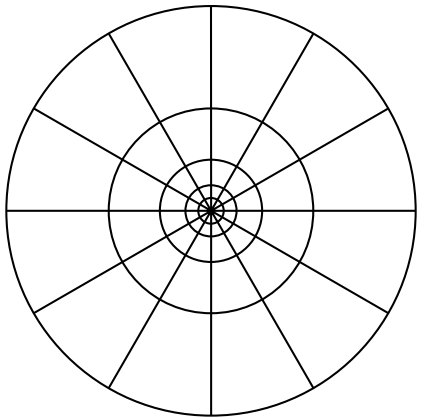
- ♢ Represent each shape as a set of sampled edge points $p_1, \ldots, p_n$
- ♢ For each point p_i, compute its *shape context*
  - a function that represents p_i's relation to the other points in the shape



Shape P



Shape Q



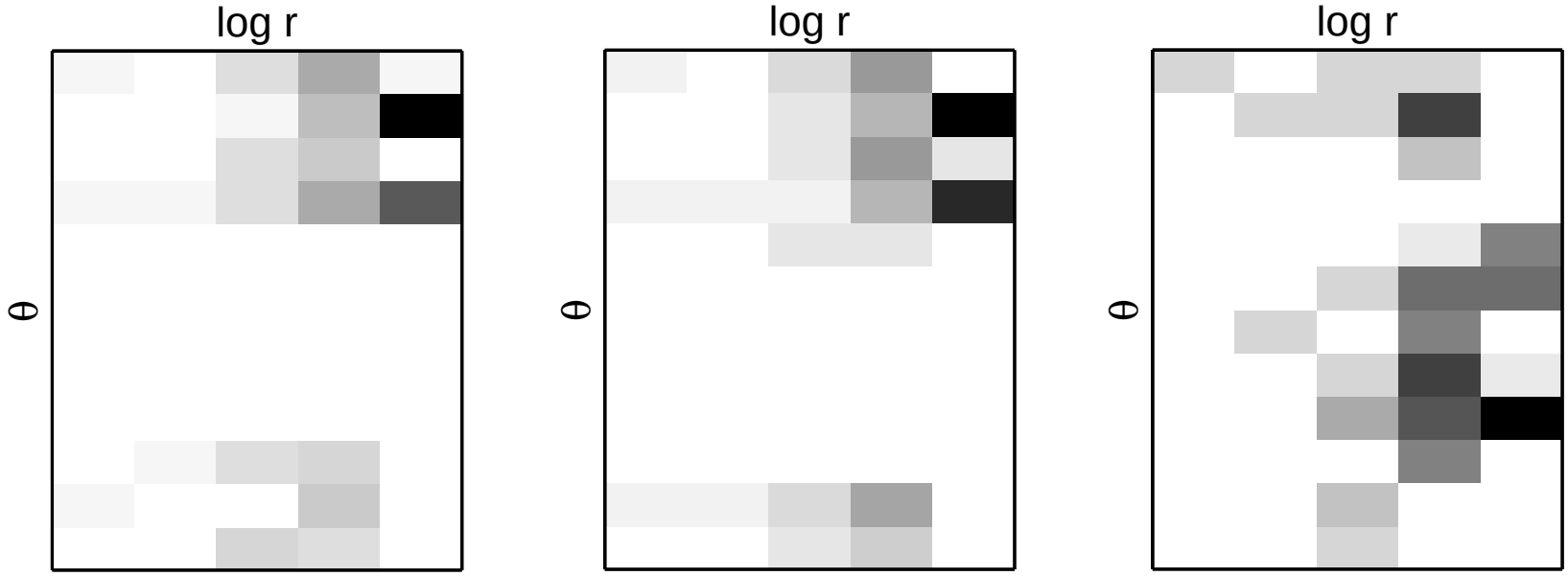
Polar coordinates $(\log r, \theta)$

Shape-context matching contd.

A point's *shape context* is a histogram:

a grid of discrete values of $\log r$ and θ

the grid divides space into "bins," one for each pair $(\log r, \theta)$



$h_i(k) =$ the number of points in the k'th bin for point p_i

Shape-context matching contd.

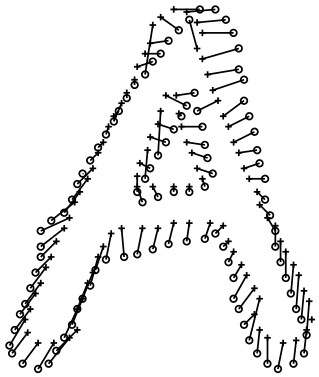
For p_i in shape P and q_j in shape Q, let

$$C_{ij} = \frac{1}{2} \sum_{k=1}^{K} [h_i(k) - h_j(k)]^2 / [h_i(k) + h_j(k)]$$

C_{ij} is the χ^2 distance between the histograms of p_i and q_j. The smaller it is, the closer the more alike p_i and q_j are.

Find a one-to-one correspondence between the points in shape P and the points in shape Q, in a way that minimizes the total cost

weighted bipartite matching problem, can be solved in time $O(n^3)$



Simple nearest-neighbor learning gives 0.63% error rate on NIST digit data

Summary

Vision is hard—noise, ambiguity, complexity

Prior knowledge is essential to constrain the problem

Need to combine multiple cues: motion, contour, shading, texture, stereo

"Library" object representation: shape vs. aspects

Image/object matching: features, lines, regions, etc.