Computer Vision, Week 11

Professor Hager
http://www.cs.jhu.edu/~hager

Outline for Today

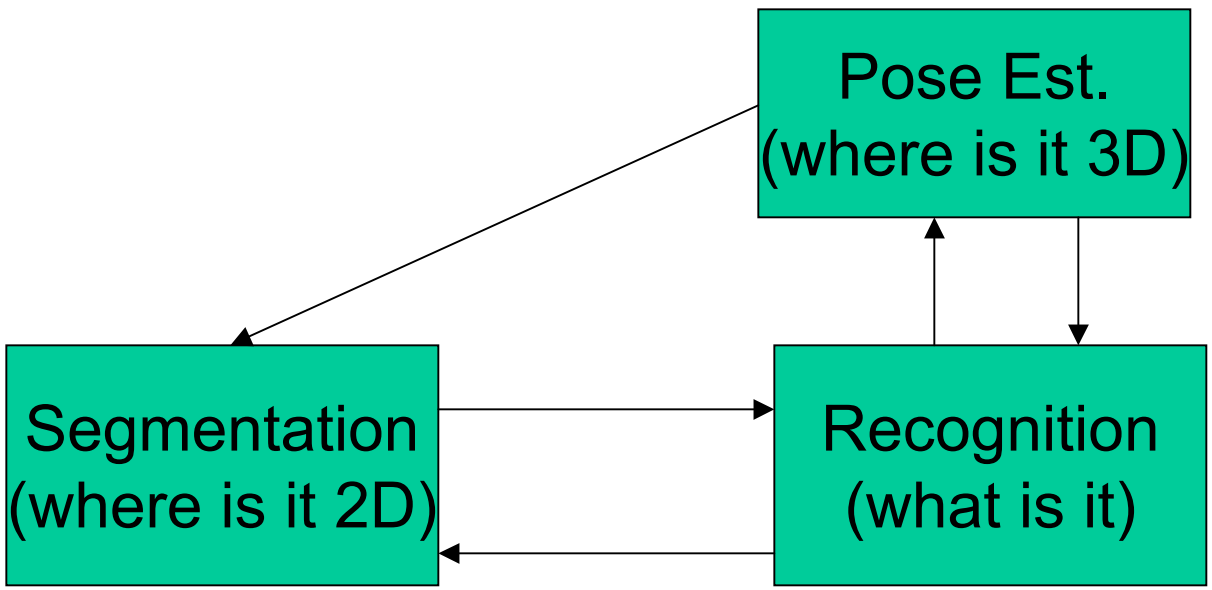
Object recognition overview

Image-based object recognition

Object Recognition: The Problem

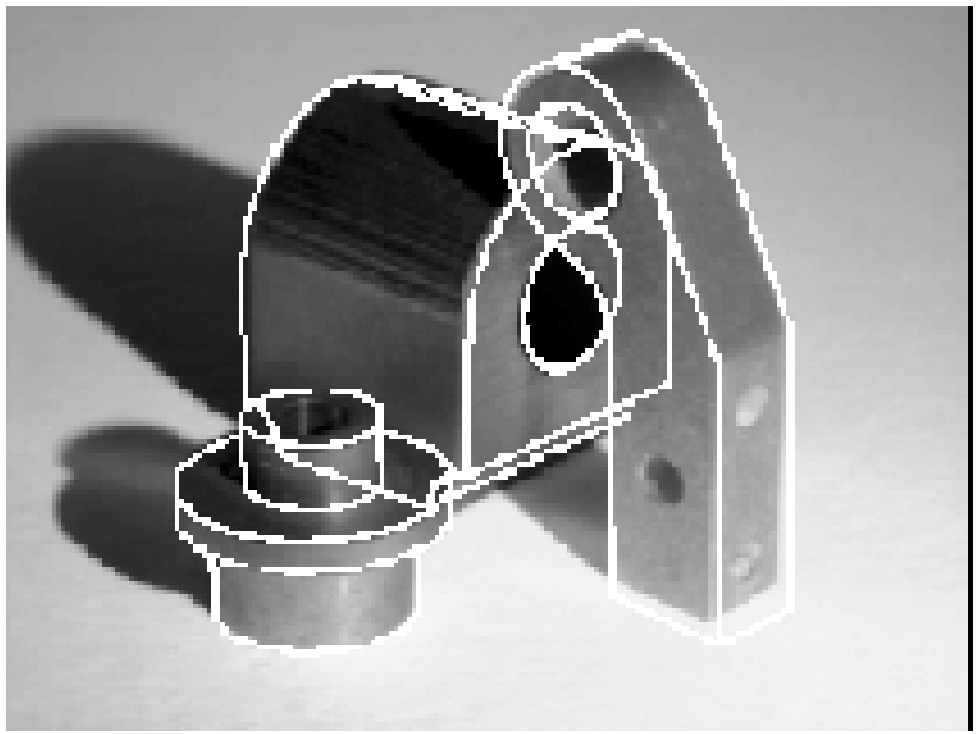
Given: A database D of “known” objects and an image I:

1. Determine which (if any) objects in D appear in I
2. Determine the pose (rotation and translation) of the object



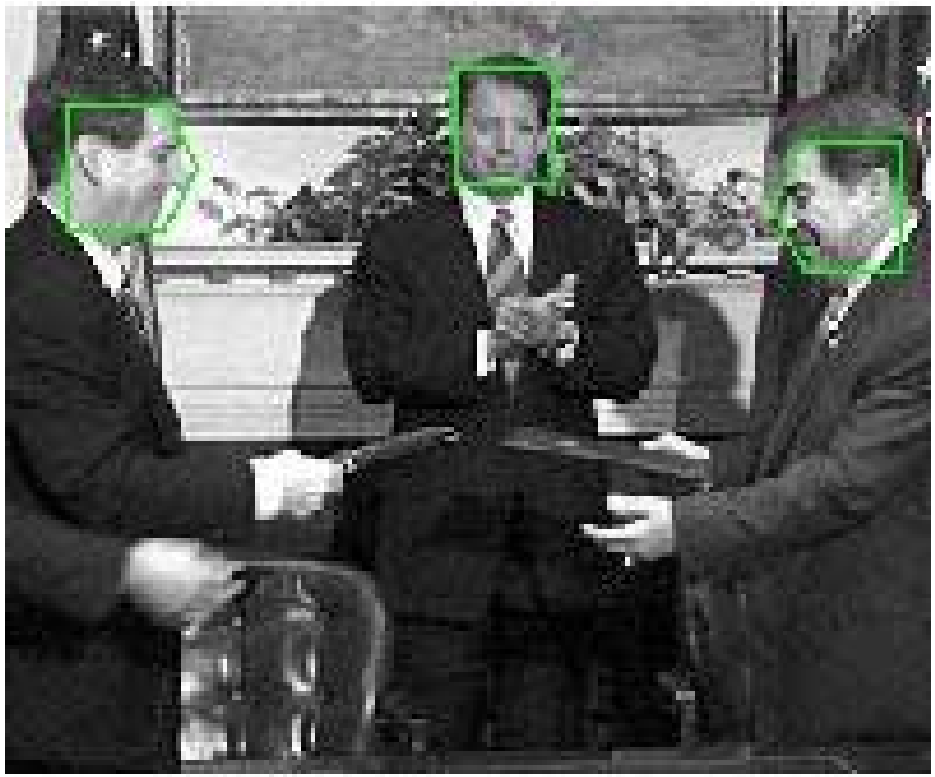
The object recognition conundrum

Problems of Computer Vision: Recognition



Given a database of objects and an image determine what, if any of the objects are present in the image.

Problems of Computer Vision: Recognition



Given a database of objects and an image determine what, if any of the objects are present in the image.

Object Recognition Issues:

- How general is the problem?
  - 2D vs. 3D
  - range of viewing conditions
  - available context
  - segmentation cues
- What sort of data is best suited to the problem?
  - local 2D features
  - 3D surfaces
  - images
- How many objects are involved?
  - small: brute force search
  - large: ??

Object Recognition Approaches

- Interpretation trees:
  - use features
  - compute “local constraints”
- Invariants:
  - use features
  - compute “global indices” that do not change over viewing conditions
- Image-based:
  - store information about every possible view

Image-based Object Recognition

An observation:

If we have seen an object from every viewpoint and under all lighting conditions, then object recognition is “simply” a table lookup in the space of 2D images

Another way to view it:

Consider an image as a point in a space
Consider now all points generated as above

Then, an object is some “surface” in the space of all images

Image-based Object Recognition

An observation:

If we have seen an object from every viewpoint and under all lighting conditions, then object recognition is simply a table lookup (given segmentation)

128x128 image = 2^{14} bytes/image
128 directions, 16 illuminants = 2^{11} cases
Therefore, 2^{25} bytes of storage: 32 Mb/object

The problem is:

Images are big

Viewing conditions are infinite

Computers are finite

Objects are surrounded by other objects

Therefore:

We need to compress the data

We need to keep the search simple

We need a means of segmenting out potential objects

Image-based Object Recognition

- How should we compare objects?
  - recall image cross-correlation

$$c(I_1,I_2) = 1/K \; sum_{i,j} \; I_1(i,j) \; I_2(i,j) = 1/K \; vec(I_1) \bullet vec(I_2)$$

- But, we don't want brightness or contrast to enter in, so define
  - $I^* = vec((I - u_I)/\|I - u_I\|)$ (think of this as a zero-mean, unit norm vector)

- And then, an interesting fact:
  - let $X = [I^*_1, I^*_2, ..., I^*_N]$
  - let e_i be the eigenvectors of XX^t (or the singular values of X)
  - then $I^*_j = sum_{j<N} \; g_{i,j} \; e_j$ where $g_{ij} = e_i \bullet I^*_j$

Image-based Object Recognition

- In practice, we don't need all of the eigenvectors (there are at most N), so
  - let $X = [I^*_1, I^*_2, ..., I^*_N]$
  - let e_i be the eigenvectors of XX^t
  - then $I^*_j \sim \text{sum}_{j<k}\, g_{i,j}\, e_j$ where $g_{ij} = e_i \bullet I^*_j$ and $k << N$

- Finally, note that (letting E be the matrix of eigenvectors)

$$\begin{aligned} \| I^*_1 - I^*_2 \| = \| E\, g_1 - E\, g_2 \| &= (E\, g_1 - E\, g_2)^t (E\, g_1 - E\, g_2) \\ &= (g_1 - g_2)^t\, E^t\, E\, (g_1 - g_2) \\ &= \| g_1 - g_2 \| \end{aligned}$$

- Thus, we can represent images in terms of a low (k) dimensional vector g

Image-based Object Recognition: Assumptions

1. Each image contains only one object
2. Objects are imaged by a fixed camera under weak perspective
3. Images are *normalized in size* so that the image is the minimum frame enclosing the object.
4. The energy of the pixel values in the image is normalized to 1.
5. The object is completely visible and unoccluded in all images.

Image-based Object Recognition: Learning

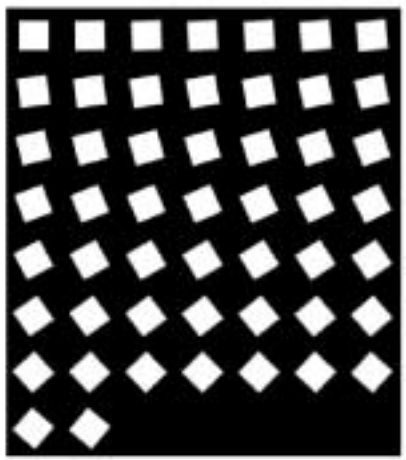
- Gather up all of the images of all objects under all viewing conditions:
  - segment to contain just the object; sample to common size
  - subtract the mean of the result from each image
  - normalize 0 mean images to unit norm
  - gather all resulting images into a matrix M (for models)

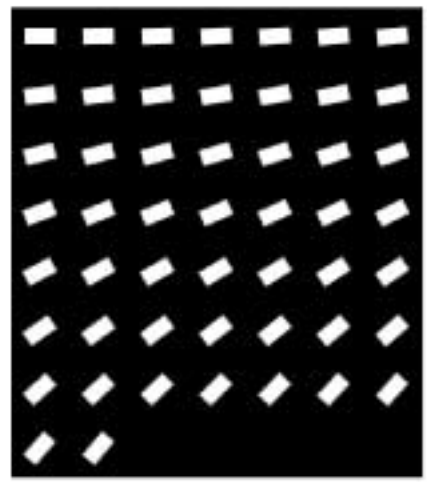
- Compute the eigenvalues and eigenvectors of $M M^t$
  - we can use SVD to do this!

- Retain the k eigenvectors with the largest associated eigenvalues
  - Usually, choose k such that $\sigma_{,k} / \sigma_{,1} < \tau$ where τ is small (e.g. .05).
  - Call the resulting matrix E (for eigenvalue projection).

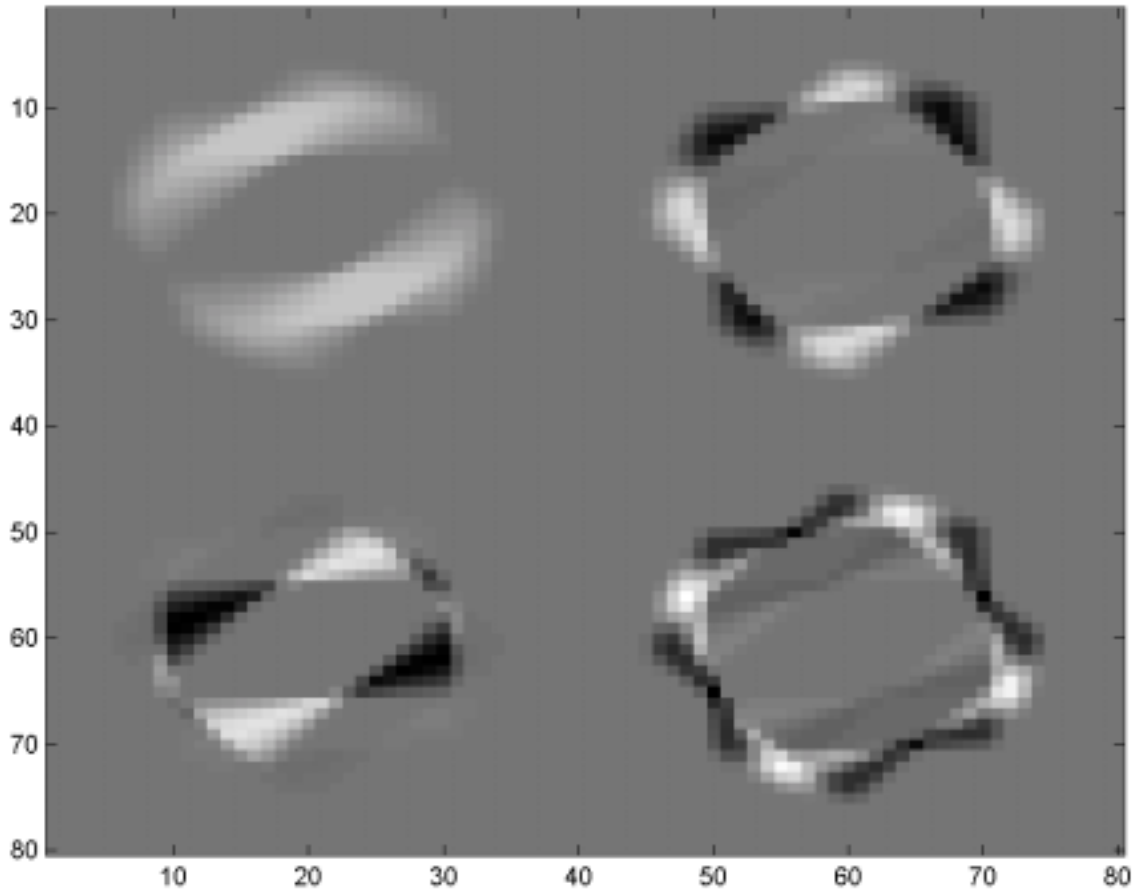
- Store a vectors $C_o = \{g^o_i = E^t I^0_i\}$ for each image i of object o

An example: input images





An example: basis images



An example: surfaces of first 3 coefficients

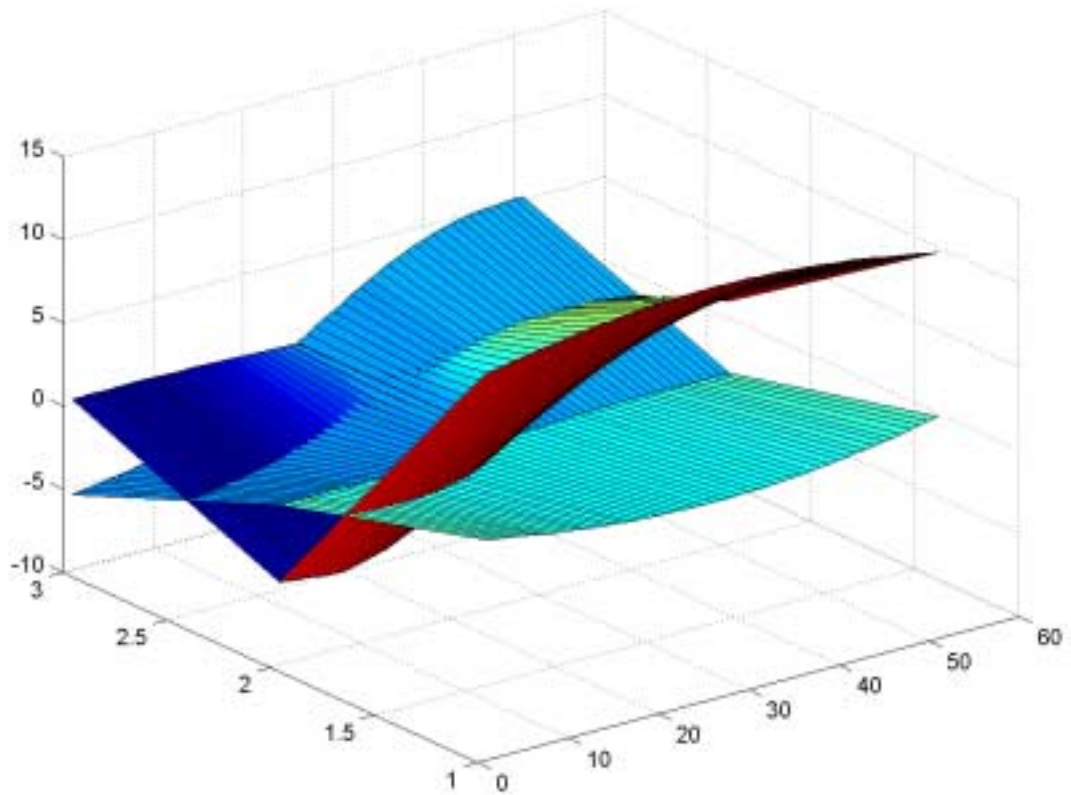


Image-based Object Recognition: Identification

- Prepare image
  - segment object from background
  - resample to be same size as model images
  - subtract *model* mean
  - normalize to unit norm
- Compute $g^* = E\ I$ where I is the result of the previous step
- Locate $\text{argmin}_o \min_{g \epsilon C} \| g - g^* \|$
  - there are faster techniques (e.g. k-d trees) for doing this
- Return O as the identification of the object
  - as a side effect, return the pose (and lighting if desired) of the object

An Example

- Columbia SLAM system:
  - can handle databases of 100's of objects
  - single change in point of view
  - uniform lighting conditions

Courtesy Shree Nayar, Columbia U.

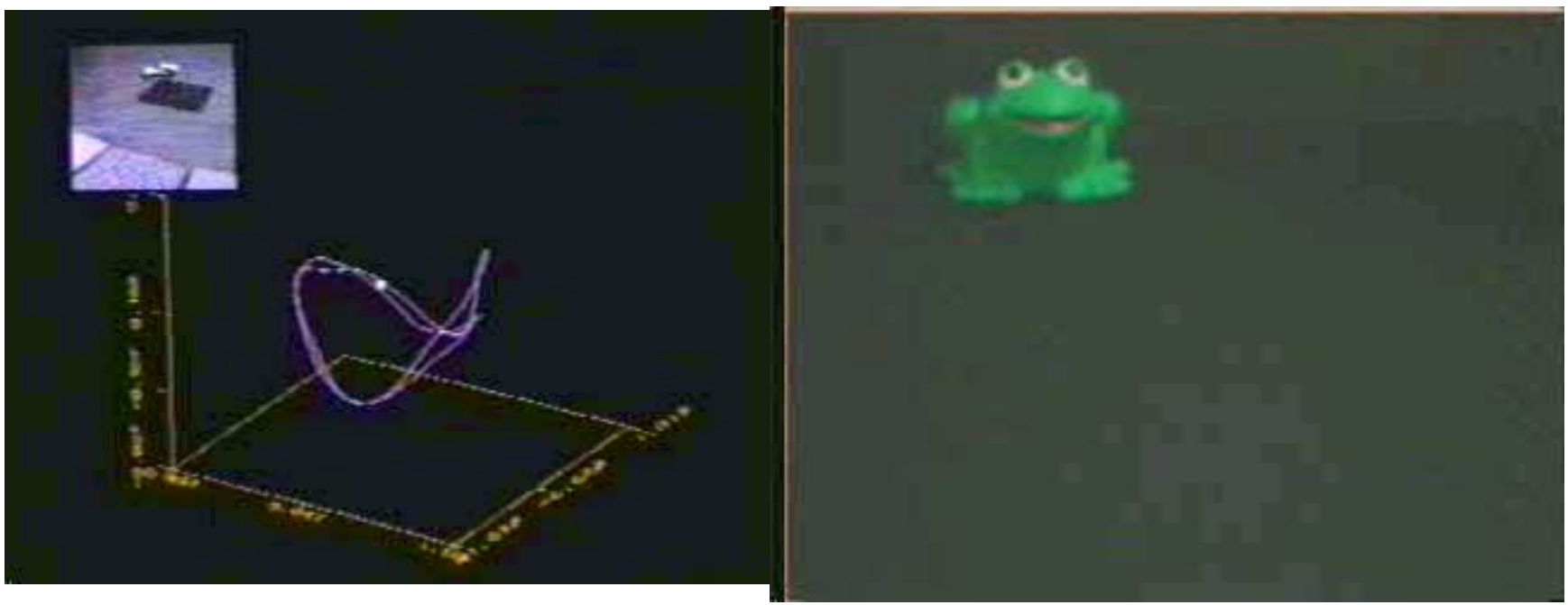


Image-based Object Recognition: Limitations

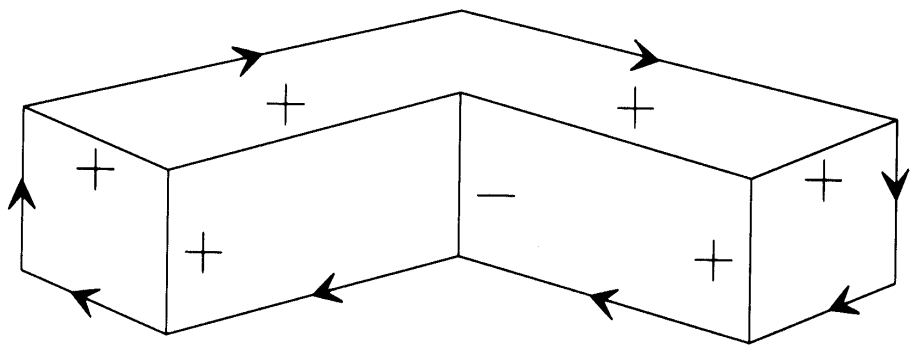
- Hard to get all of the samples needed.
- Better for Lambertian; less so for specular objects
- Assumes a constant background or good segmentation

Constraint-Based Approaches

- Use constraints available on image features to recognize it
- A good starter is the Huffman and Clowes line interpretation algorithm:

We Interpret Line Drawings As 3D

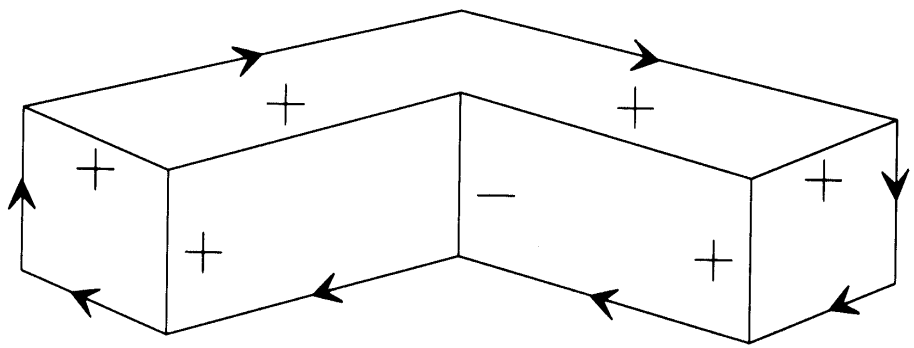
- We have strong intuitions about line drawings of simple geometric figures:
  - We can detect possible 3D objects (although our information is coming from a 2D line drawing).
  - We can detect the convexity or concavity of lines in the drawing.
  - If a line is convex, we have strong intuitions about whether it is an occluding edge.



(liberally borrowed from Embick and Marcus)

Convexity Labeling Conventions

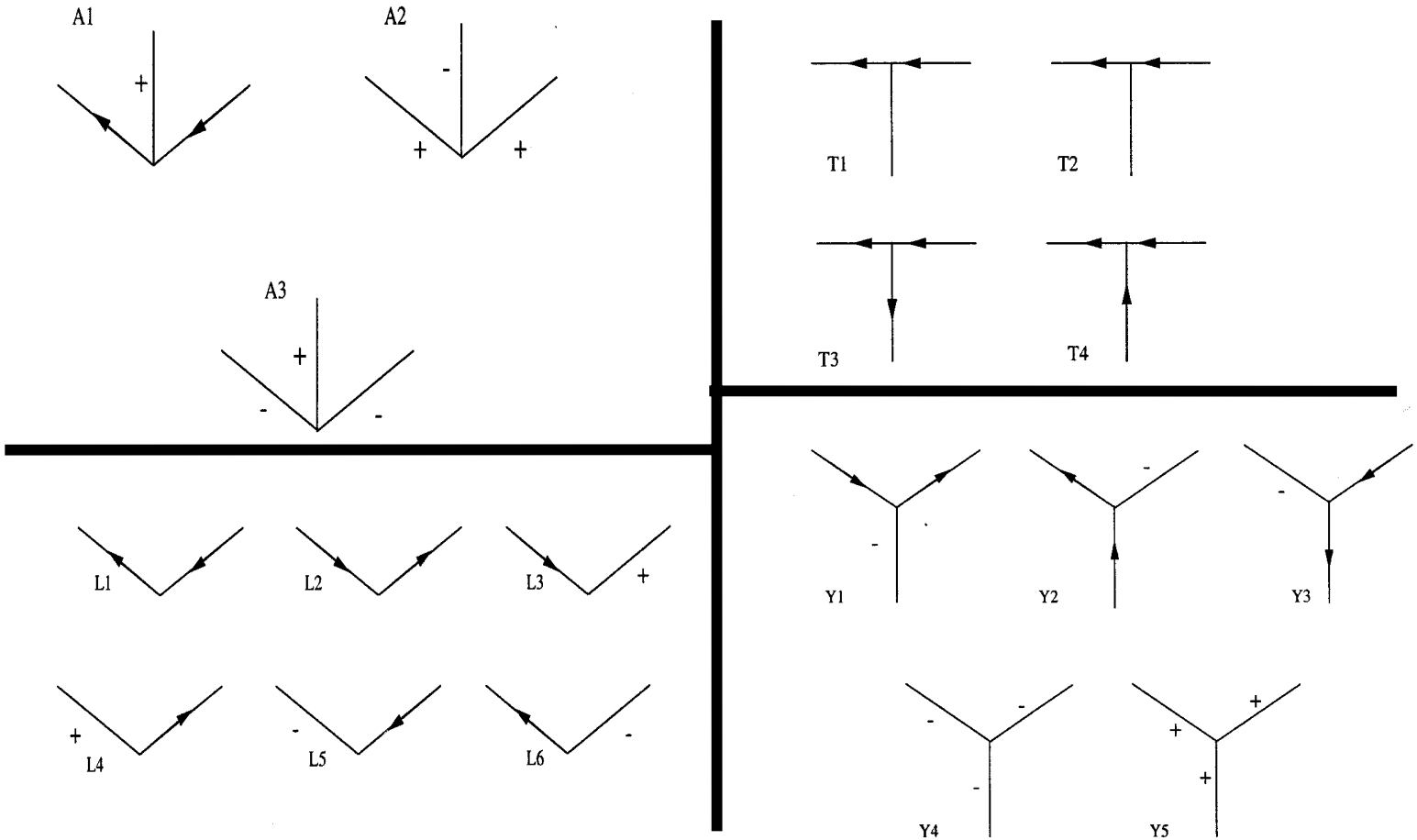
1. A line labeled ***plus (+)*** indicates that the corresponding edge is ***convex*** ;
2. A line labeled ***minus (-)*** indicates that the corresponding edge is ***concave***;
3. An ***arrow*** indicates an ***occluding*** edge. To its right is the body for which the arrow line provides an edge. On its left is space.



(liberally borrowed from Embick and Marcus)

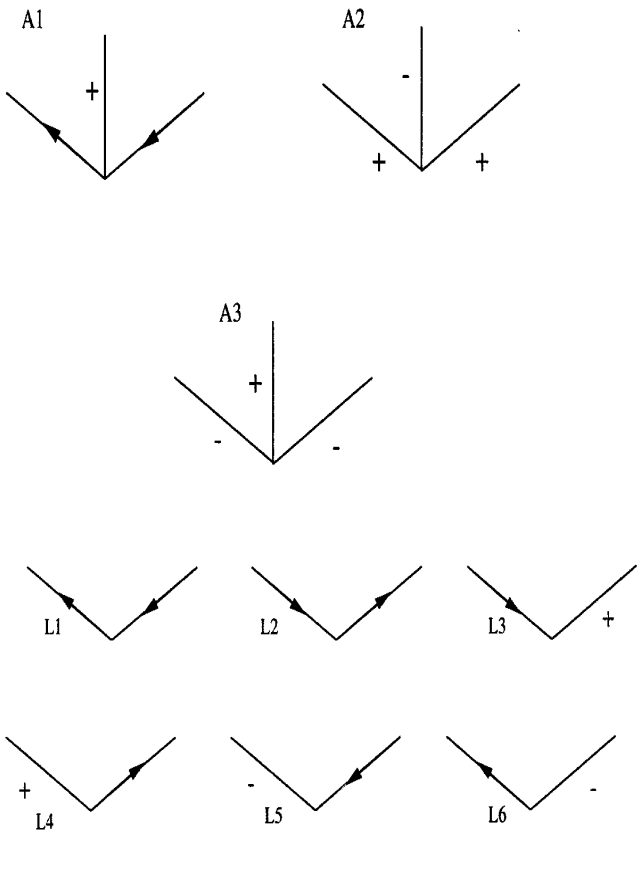
Edge Consistency

Any ***consistent assignment*** of labels to the junctions in a picture must assign the ***same*** line label to any given line.

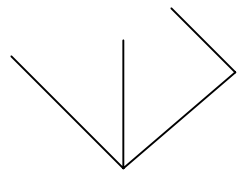


(liberally borrowed from Embick and Marcu

An Example of Edge Consistency



- Consider an arrow junction with an L junction to the right:



- ***A1 and either L1 or L6 are compatible*** since they both associate the same kind of arrow with the line.
- ***A1 and L2 are incompatible***, since the arrows are pointed in the opposing directions,
- ***Similarly, A1 and L3 are incompatible.***

(liberally borrowed from Embick and Marcus)

The Generate and Test Algorithm

1. ***Generate*** all conceivable labelings.
2. ***Test*** each labeling to see whether it violates the edge consistency constraint; throw out those that do.

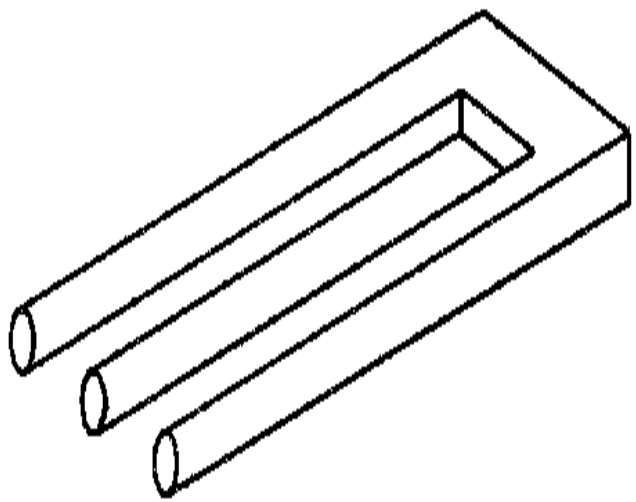
BUT:

- Each junction has on average ***4.5*** labeling interpretations.
- If a figure has ***N*** junctions, we would have to generate and test 4.5^N different labelings.
- Thus the Generate and Test Algorithm is $O(4.5^N)$.

(liberally borrowed from Embick and Marcus)

Is $O(4.5^N)$ Bad??

- A picture with 27 junctions, like the devil's trident) will not be rejected until all 4.5^{27} hypotheses have been rejected, leaving no interpretation.
- $4.5^{27} = 433249302231073824.244664378464222$
- A computer capable of checking for edge consistency at a rate of ***1 hypothesis per microsecond*** would take about ***1 million years*** to establish that the devil's trident has no consistent interpretation!



(liberally borrowed from Embick and Marcus)

Locally Consistent Search

- The Generate and Test algorithm considers too many hypotheses that are ***simply outright impossible*** according to the ***edge consistency constraint***.
- A better algorithm would ***exploit locally consistent labelings*** to construct a ***globally consistent interpretation***:
  1. No junction label would be added unless it was consistent with its immediate neighbors;
  2. If the interpretation could not be continued because no consistent junction label can be added, that interpretation would be abandoned immediately.

(liberally borrowed from Embick and Marcus)

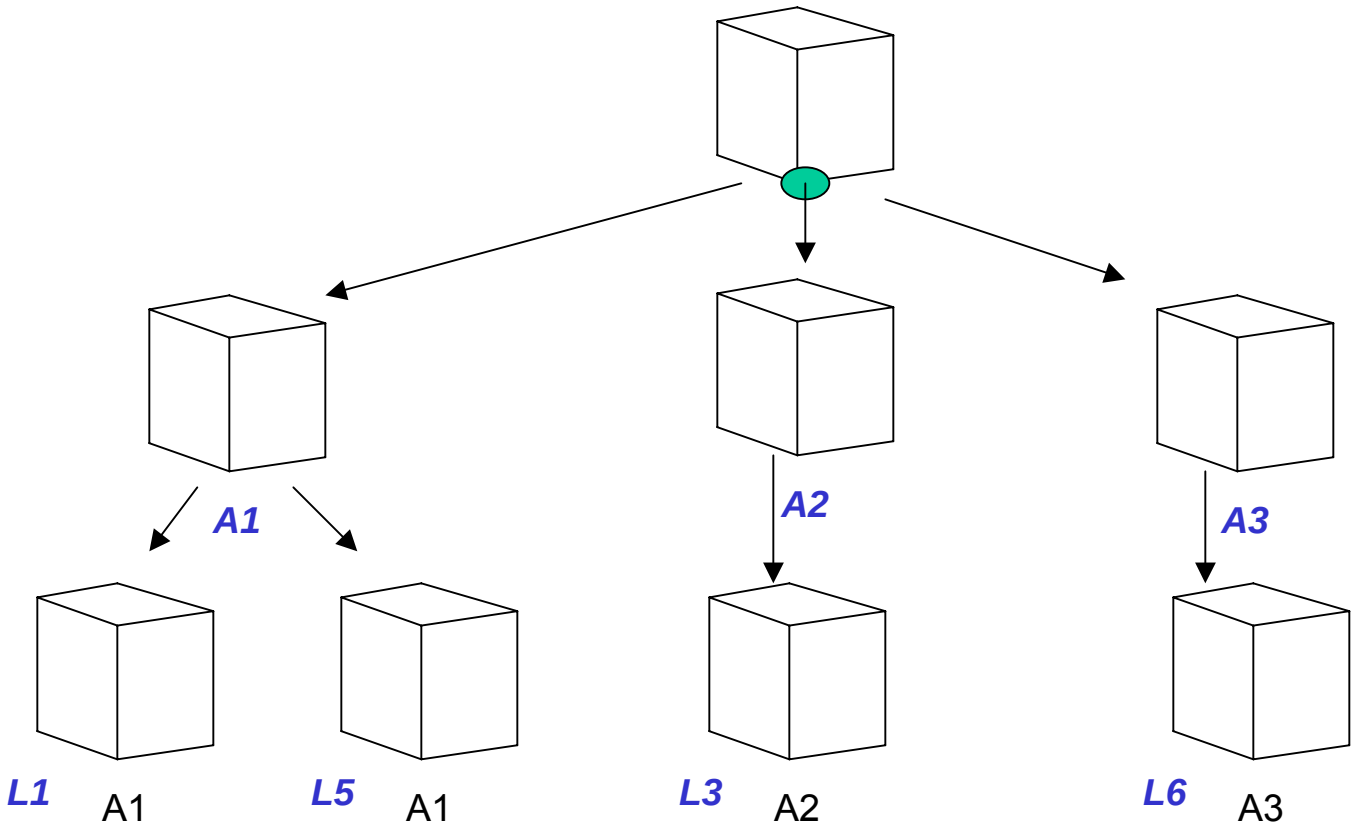
Search Trees to the Rescue!

We could implement this by constructing a ***search tree***:

1. Selecting some junction in the drawing as the ***root***.
2. Label children at the ***1st level*** with ***all*** possible interpretations of that junction.
3. Label their children with possible ***consistent*** interpretations of some junction adjacent to that junction.
4. Each level of the tree adds one more labeled node to the growing interpretation.
5. ***Leaves*** represent either ***futile*** interpretations that cannot be continued or ***full*** interpretations of the line drawing.

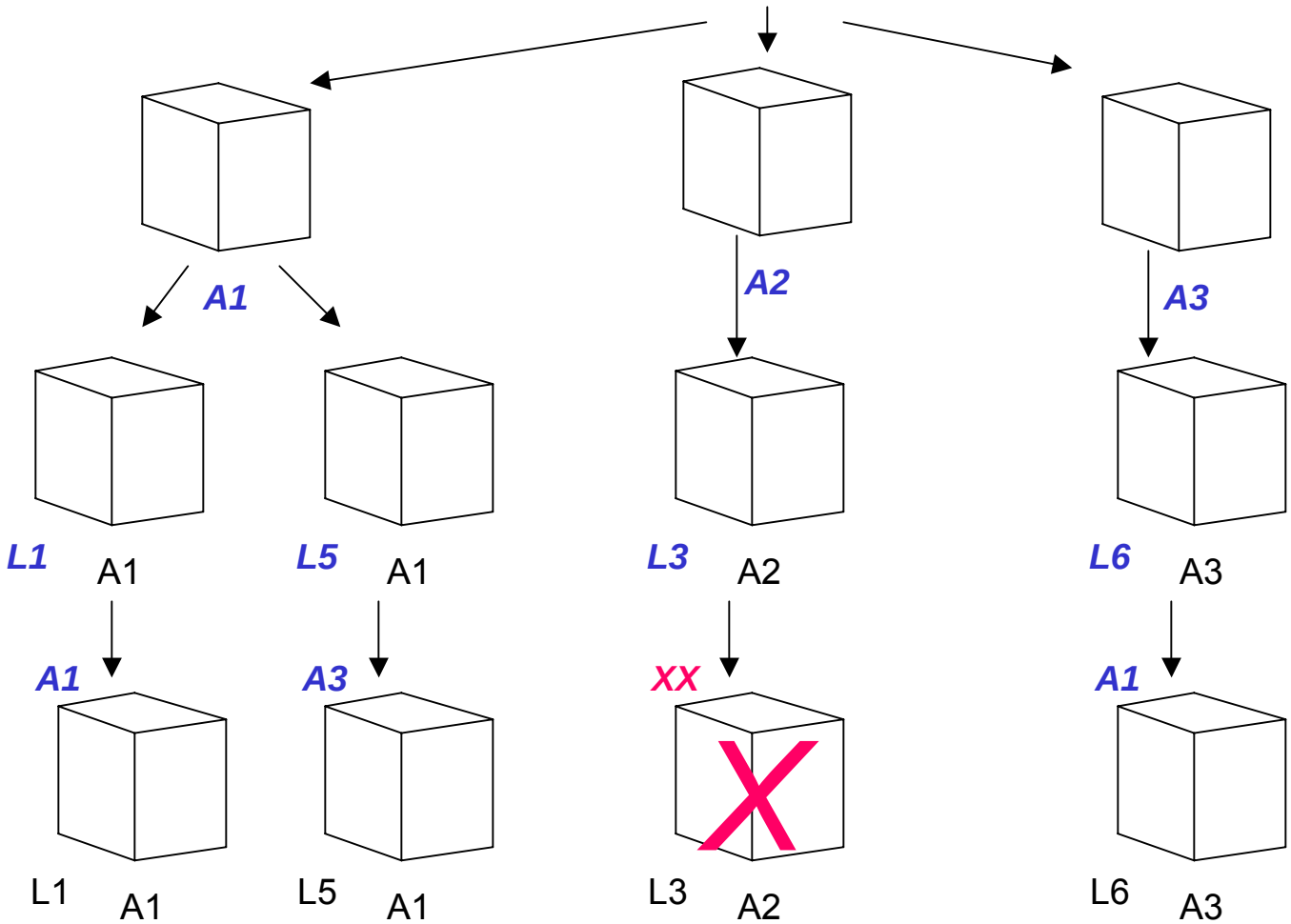
(liberally borrowed from Embick and Marcus)

The Top Three Levels of a Search Tree



(liberally borrowed from Embick and Marcus)

The Fourth Level of the Search Tree



(liberally borrowed from Embick and Marcus)

Breadth-First Search

- In ***breadth-first search***, all nodes at the 1st level of the tree are generated, ***then*** all nodes at the 2nd, ***then*** the 3rd level, and so on until the leaves are reached.
- If the ***average branching factor*** (the average number of children per node) is ***B*** and the depth of the tree is ***N***, then the time complexity of the algorithm is

$$T=O(B^N).$$

- We have to keep track of the whole tree as we search, so the space complexity is also

$$S=O(B^N).$$

(liberally borrowed from Embick and Marcus)

Depth-First Search

- ***Depth-first search*** follows one branch of the tree down to its leaf and then ***backtracks*** to follow the other branches down to their leaves.
- Again, if the average branching factor is ***B*** and the depth of the tree is ***N***, then the time complexity of the algorithm is still

$$T=O(B^N).$$

- But it requires ***less space*** since we only have the ***hold the junctions on one path from the root*** with information about which ones were ***choice points***. So its space complexity is only

$$S=O(N).$$

(liberally borrowed from Embick and Marcus)

Constraint Propagation

Waltz's insight:

- Pairs of ***adjacent*** junctions (junctions connected by a line) ***constrain*** each other's interpretations!
- These ***constraints*** can ***propagate*** along the connected edges of the graph.

Waltz Filtering:

- Suppose junctions *i* and *j* are connected by an edge. ***Remove any labeling from i that is inconsistent with the labeling assigned in j.***
- By ***iterating*** over the graph, the constraints will ***propagate***.

(liberally borrowed from Embick and Marcus)

Limitations of this Idea

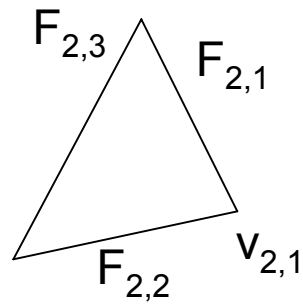
- Assumes that we have polygonal objects
- Assumes contours only come from such objects
- Assumes contours are complete
- Generic objects (but commensurate loss of power)

Interpretation Trees: Basic Idea

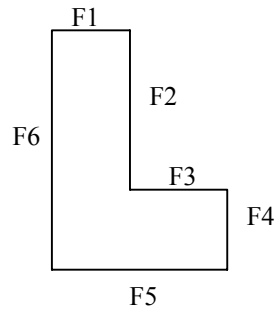
- Given:
  - A (usually 3D geometric) model, we a set of features and defined relationships between features F_1, F_2, ... F_n
    - unary: e.g. length range
    - binary: e.g. distance range
    - trinary: e.g. angle between triples of points
  - An observed set of features f_1, f_2, ... f_m
- Compute:
  - all possible matches between model features and observed features which respect the given constraints.

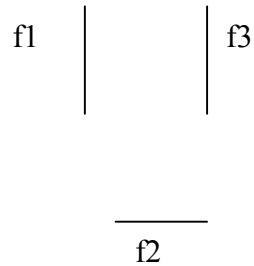
Constraints: Examples

- Length (line)
  - $| F_{2,2}| = d_{2,2}$, ...
- Distance (line to line)
  - |F_{2,2} – F_{2,1}| = [dmin_{2,1},dmax_{2,1}]
- Distance (vertex to line)
  - |F_{2,3} – v_{2,1}| = [dmin_{2,3},dmax_{2,3}]
- Angle (line to line)
  - |F_{2,2} . F_{2,1}| = a_{2,1}
- Area (face)
  - .....



angles	F1	F2	F3	F4	F5	F6
F1	0	3π/2	0	3π/2	π	π/2
F2	π/2	0	π/2	0	3π/2	π
F3	0	3π/2	0	3π/2	π	π/2
F4	π/2	0	π/2	0	3π/2	π
F5	π	π/2	π	π/2	0	3π/2
F6	3π/2	π	3π/2	π	π/2	0





angles	f1	f2	f3
f1	0	π/2	π
f2	3π/2	0	π/2
f3	π	3π/2	0

distance	F1	F2	F3	F4	F5	F6
F1	[0,1]	[0,5]	[4,8]	[5,13]	[9,13]	[0,10]
F2	[0,5]	[0,4]	[0,5]	[1,10]	[1,10]	[1,10]
F3	[4,8]	[0,5]	[0,1]	[0,2]	[1,5]	[1,8]
F4	[5,13]	[1,10]	[0,2]	[0,1]	[0,5]	[4,13]
F5	[9,13]	[1,10]	[1,5]	[0,5]	[0,4]	[0,13]
F6	[0,10]	[1,10]	[1,8]	[4,13]	[0,13]	[0,9]

distance	f1	f2	f3
f1	[0,1]	[1,5]	[1,2]
f2	[1,5]	[0,1]	[1,5]
f3	[1,2]	[1,5]	[0,1]

Interpretation Tree: a 2D Example