**Problem:**

Build a machine that can visually interpret the world (like you do!).

But what does it mean to interpret the world?

**Inputs:**

- a "visual" scene (the images that fall on your retinae, for humans, or the CCD chip, for digital cameras)
- some kind of knowledge about the objects in the world, e.g., what you've seen before, know about, etc.

**Output:**

- identity and location of objects in the scene

**How does this relate to 228?**

- You have a query: What object(s) am I looking at?

- You have a BIG file: Your visual memory (brain) is a bunch of records, one per object (adults have knowledge of approximately 10,000 objects).

- You have a search task: From the file, search for the record that contains the object I'm looking for.

**Challenges:**

- How do you represent objects?, i.e., what is the record structure?
- How do you find the record you want? Sequential processing?
- If you want to do indexing on a sorted file, how could you sort the objects? How can you sort by geometry? Is $\log_2 10{,}000$ acceptable?
- What other method do we have for locating a record?

## Hashing

- What do we hash with?

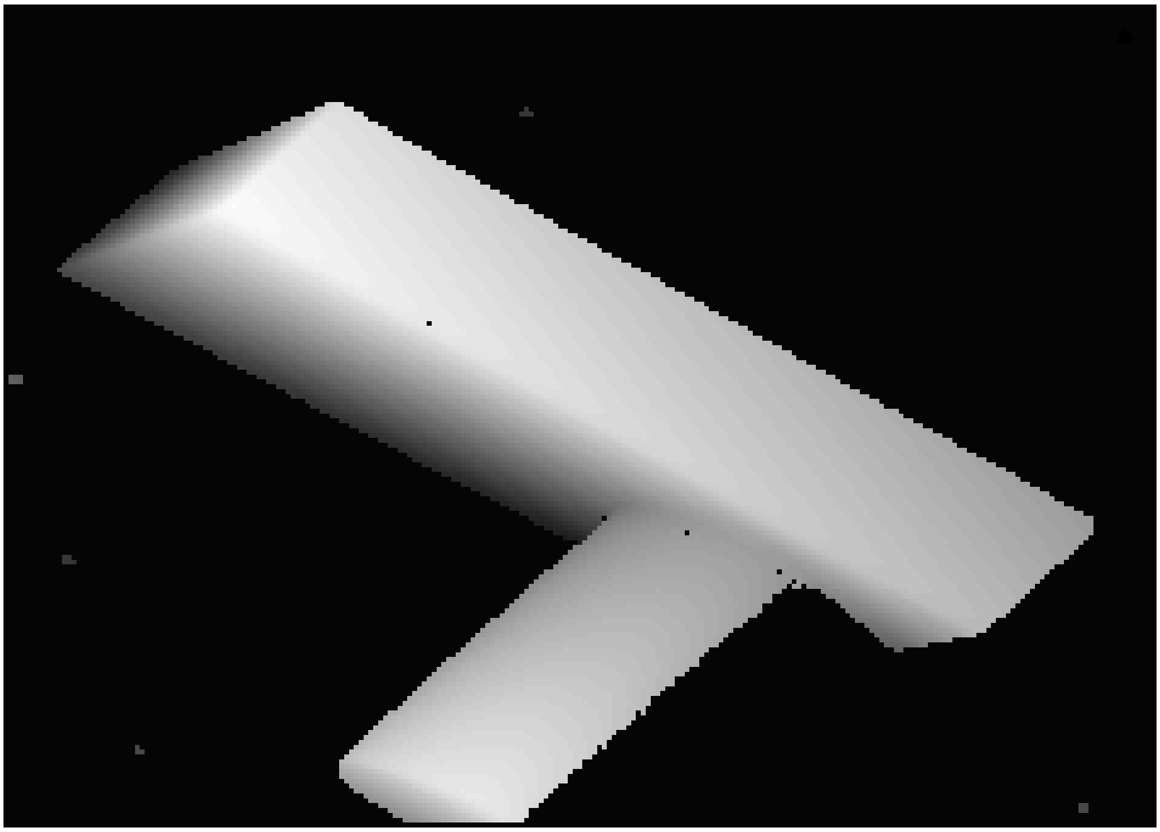
- Are the features stable? Invariant to scale, translation, rotation, object articulation, occlusion, viewpoint?

- Once you get to the right bucket, how do you “find” your “secondary key (query)”?

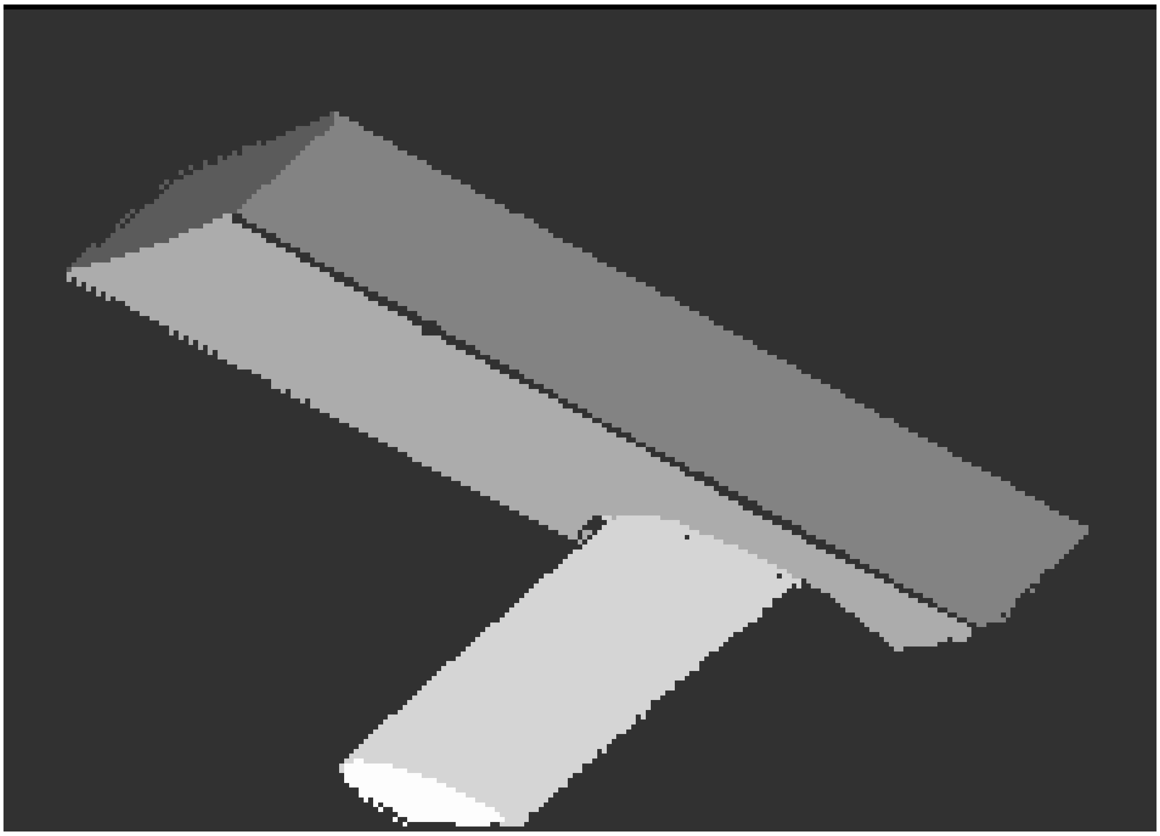
## Flynn and Jain

- takes a 3-D range image as input.
- "carves up" the input image into a set of regions (or patches), each representing a planar, spherical, or cylinderical surface patch.
- extracts invariant features from the patches. Any ideas?
- the invariant features are used to compute a hash address for a table whose entries contain objects whose invariant features are similar.
- models receiving lots of "support" must be verified before identity can be extablished.

Range Image:



Region Segmented Image:



What are the invariant features?

- plane-plane
- plane-cylinder
- plane-sphere
- cylinder-sphere
- cylinder-cylinder
- sphere-sphere

Step 1: Hash Table Construction

1. Choose a triple of surfaces in synthetic range image of an object.

2. There will be 10 separate hash tables, corresponding to different types of features. Let's look at an example.

3. (plane, cylinder, sphere): We will compute the feature between the plane and cylinder to get one value, and the features between cylinder and sphere to get a second value.

4. These two values will give us a location in a 2-D table. At that location will be a pointer to all models that have 3 surfaces in that particular configuration, along with which particular model surfaces we're talking about.

5. Another example: (plane, sphere, sphere): Compute two features: the plane and the first sphere, and the plane and the second sphere. What if we swapped the two spheres? We'd get different hash values. Therefore, we need to put the same model in two buckets.

## Step 2: Using the Hash Table at Run Time

1. Find all triples of surfaces in the “query” image.

2. For each triple, hash to a location in the appropriate table to get object hypothesis.

3. Verify the hypothesis. How? Three-surface correspondence allows us to compute the position and orientation of the model w.r.t. the image. Then synthesize an image of the object at that “pose” and compare with the original image.

4. If we agree, we’ve recognized the object!!

How do we deal with “noise”? What is noise?

Flynn and Jain hash with +/- 5deg of the feature value to “cover” nearby locations. This means that objects with similar shape should live in nearby buckets!