GEOMETRIC HASHING: AN OVERVIEW
What is geometric hashing good for?

- Identification of test objects from a set of known objects using 2D or 3D features
- Object models can be made invariant to translation, rotation, scaling, affine and projective transformations! NEAT!
- Uses an arbitrary feature detector. Works with SIFT etc.
- Quick model identification.
Building Object Models

- Extract Features
- Define a basis with respect to two features
- Parameterize features with respect to basis.
- Add the parameterized features to a 2D hash table
- Repeat with a new basis until every basis is added
- Do some hash table tweaking to improve worst case performance.
- Do further hash table tweaking to improve performance in the case of noise.
Procedure for Model Detection.

- **Model Detection**
  - Extract features from the image and parameterize them with respect to a basis.
  - Do a hash table lookup for each feature. Hash table entries vote for the model.
  - For each candidate model complete the necessary transformation, and test against the input.
  - If the match isn’t good pick a new basis and restart the procedure.
Feature Extraction
Scale and Create a Basis

Scale to unit length (scale invariance)

Rotate to +x axis (rotation invariance)
We also create a 2D hash table

- from \( u \) from \(-1\) to \(1\)
- from \( v \) from \(-1\) to \(1\)

Model and basis:
- Kitty(4,1)
- Homer(1,2)
- Zim (2,7)
- Mickey(3,1)
Now repeat for each basis

Each arrow is a basis
Recognition of Objects

- Find Features
- Select Basis
- Parameterize Features
- Do hash table lookup
- Histogram number of matches per model
Recognition of Objects

Potential Matches

Threshold

(3,2)  (3,1)  (4,1)  (1,4)
Recognition of Objects

Verification

Least Square Transform Recovery
(find the rotation, and translation of the basis)
Geometric hashing has a number of nice properties in 2D

<table>
<thead>
<tr>
<th>Transformation Recovered:</th>
<th>What is needed:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translation</td>
<td>One point centered at origin</td>
</tr>
<tr>
<td>Translation &amp; Rotation</td>
<td>Two points</td>
</tr>
<tr>
<td>Translation, Rotation, Scaling</td>
<td>Two points and scale to length of one.</td>
</tr>
<tr>
<td>Affine Transformation</td>
<td>Three Points</td>
</tr>
<tr>
<td>Projective Transformation</td>
<td>Four Points</td>
</tr>
</tbody>
</table>
**Geometric Hashing also works in 3D**

<table>
<thead>
<tr>
<th>Transformation Recovered:</th>
<th>What is needed:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translation</td>
<td>One point centered at origin</td>
</tr>
<tr>
<td>Translation &amp; Rotation</td>
<td>Three points –or– Two non co-linear vectors</td>
</tr>
<tr>
<td>Translation, Rotation, Scaling</td>
<td>A line and a point or Two non-collinear non-coplanar lines</td>
</tr>
</tbody>
</table>
Run time of Geometric Hashing

$M =$ number of models
$n =$ number of features per model.
$c =$ the number of elements in a basis
$H =$ hash table access time.
$S =$ number of features in a test image

Computing the hash table runs at $O(Mn^{c+1})$
Looking up a model in the hash table takes $O(HS^{c+1})$

The run time is not great, but tolerable when $H$ is small as $H$ can vary i.e. $O(1) \leq H \leq O(Mn^{c+1})$
What if we end up with a really bad hash table?

We can calculate \((f(u,v))\) or model \((f^*(u,v))\) the probability distribution function of the entries of the hash table.

Really bad hash table. Worst case performance is horrible!

We can calculate \((f(u,v))\) or model \((f^*(u,v))\) the probability distribution function of the entries of the hash table.
Rehashing a Table

Find a function $h(u,v)$ that can apply to each hash entry to get a nice uniform distribution. This is to say $h(u,v)$ makes $f(u,v)$ or $f^*(u,v)$ into an uniform distribution.

$$f(u,v) = \frac{12}{\pi} \cdot \frac{1}{(4(u^2+v^2)+3)^2}.$$ 

$$h(u,v) = \left(1 - \frac{3}{4(u^2+v^2)+3}, \tan 2(v,u)\right)$$
Intuition behind rehashing

Consider a 1D hash table.

We determine \( f(u) \) for this table

And find \( h(u) \) that does this:
Modeling and Dealing with Noise

Some noise tolerance is built into the hash table by quantization. Bigger bin sizes result in less noise but lower fidelity. One approach is to weight bins in a neighborhood or weighting the votes of each bin.
More about image noise

1. The larger the separation of basis points the smaller the effect of noise spread in the hash table. (Noise is proportionally smaller)

2. The closer a point is to the center of the basis the smaller the spread of noise in the hash table.

3. Short hash table entries contain more information than long ones.
We can construct a Bayesian formulation of geometric hashing

\[ x \ y = \text{value of basis transformation} \]

\[ u \ v = \text{hash table bin value} \]

\[ p_\mu \ p_\nu = \text{trial basis} \]

\[ \sigma = \text{error distribution} \]

\[ S = \text{number of points in scene} \]

\[ z \] is the neighborhood function for weighting hash table entries

\[
\tau = (4(x^2 + y^2) + 3) \sigma^2
\]

\[
z = \log \left(1 + \frac{(4(u^2 + v^2) + 3) \|p_\mu - p_\nu\|^2}{12S\tau} \cdot \exp \left(\frac{-(\|u,v\| - \|x,y\|^2)}{\tau / \|p_\mu - p_\nu\|^2}\right)\right)
\]

(5)