Micro Perceptual Human Computation for Visual Tasks

*ACM Transactions on Graphics*

Yotam Gingold  
*Columbia/Rutgers*

Ariel Shamir  
*Herzliya IDC*

Daniel Cohen-Or  
*Tel-Aviv University*
Historical Digression

[David Alan Grier 2005]

1700’s
Historical Digression

[David Alan Grier 2005]

1700’s

Clairaut
Historical Digression

[David Alan Grier 2005]

1700’s

Clairaut

Halley’s Comet
Historical Digression

[David Alan Grier 2005]

1800’s
Historical Digression
[David Alan Grier 2005]

1800’s

Babbage
Historical Digression
[David Alan Grier 2005]

1800’s

Babbage

Difference Engine
Historical Digression

[David Alan Grier 2005]

1900’s
Historical Digression

[David Alan Grier 2005]

1900’s

WPA/war effort/NACA
Historical Digression

[David Alan Grier 2005]

1900’s

WPA/war effort/NACA

Trinity
Historical Digression

[David Alan Grier 2005]

1900’s

ENIAC
Electronic

- Fast
- Deterministic
- Arithmetic

Human

- Slow
- Inconsistent & noisy
- ???
The Human Advantage

• Perception
• Preference
• Creativity
Human Computation

• Luis von Ahn’s 2005 PhD thesis:
  – “We treat human brains as processors in a distributed system, each performing a small part of a massive computation.”
  – “We argue that humans provide a viable, under-tapped resource that can aid in the solution of several important problems in practice.”
Example 1:

1. You and a partner see the same image.

2. Each of you must guess what words your partner is typing.

[von Ahn and Dabbish 2004]
Example 2: LabelMe

[Russel et al. 2005/2008]
Example 3:

**Soylent**

A Word Processor with a Crowd Inside

Soylent is a crowd-powered interface: one that embeds workers from Mechanical Turk into Microsoft Word.

[Join the Beta]

[Bernstein et al. 2010]
def get_normals_for_locations(image_path, locations):
    '''
    Given an 'image_path' and an iterable collection of integer (row,col)
    locations at which we want to know the normal 'locations',
    returns a list of (x,y,z) unit normals corresponding to each element
    in 'locations'.
    '''
    import oracle_normals.knowledge
    K = oracle_normals.knowledge.KnowledgePairChecking()
    K.want_to_know(image_path, locations)
    normals = K.get_answer_at_rows_cols(image_path, locations)
    assert len(normals) == len(locations)
    return normals

def generate_surface_from_normals(rows, cols, locations2normals):
    '''
    Returns a 2D array with shape( 'rows', 'cols' ) whose values are created
    by interpolating the normals given by 'locations2normals', a dictionary
    mapping integer ( row, col ) to ( x,y,z ) values.
    '''

Computation

user

application
code

electronic processors
def get_normals_for_locations(image_path, locations):
    '''
    Given an 'image_path' and an iterable collection of integer (row,col)
    'locations' at which we want to know the normal 'locations',
    returns a list of (x,y,z) unit normals corresponding to each element
    in 'locations'.
    '''
    import oracle_normals.knowledge
    K = oracle_normals.knowledge.KnowledgePairChecking()
    K.want_to_know(image_path, locations)
    normals = K.get_answer_at_rows_cols(image_path, locations)
    assert len(normals) == len(locations)
    return normals

def generate_surface_from_normals(rows, cols, locations2normals):
    '''
    Returns a 2D array with shape( 'rows', 'cols' ) whose values are created
    by interpolating the normals given by 'locations2normals', a dictionary
    mapping integer (row, col) to (x,y,z) values.
    '''

human computation

user

application

code

electronic processors

human processors
Why?

user

application

code

electronic processors

human processors

def get_normals_for_locations(image_path, locations):
    '''
    Given an 'image_path' and an iterable collection of integer (row,col)
    locations at which we want to know the normal 'locations',
    returns a list of (x,y,z) unit normals corresponding to each element
    in 'locations'.
    '''
    import oracle_normals.knowledge
    K = oracle_normals.knowledge.KnowledgePairChecking()
    K.want_to_know(image_path, locations)
    normals = K.get_answer_at_rows_cols(image_path, locations)
    assert len(normals) == len(locations)
    return normals

def generate_surface_from_normals(rows, cols, locations2normals):
    '''
    Returns a 2D array with shape( 'rows', 'cols' ) whose values are created
    by interpolating the normals given by 'locations2normals', a dictionary
    mapping integer ( row, col ) to ( x,y,z ) values.
    '''
    # Interpolate normals to generate surface values.
Why?

• Make the impossible possible
Why?

• Make the impossible possible
• Speed and cost
Humans using Computers
Range of Solutions

- How much human and how much computer is involved?

- More Human “cycles”
- More Computer “cycles”
- Fully Automatic (no human)
- Human Computation
- Interactive Application
- Let a human do it
Type of Human Cycles

• You can also think of the type of activity the human does.
Algorithm Design Pattern

Large Task → Partition → Micro-task → Verify → Compose
The Question We Ask

• What is the minimum amount of information a human could provide in order to solve the original problem?

• Rephrase the algorithm in terms of the smallest piece of information that without it the problem could not be solved.
Three Example Algorithms

• Given an image, create
  – depth layers
  – a normal map
  – a bilateral symmetry map
Issues

• Motivation:
  – Money: Amazon’s Mechanical Turk
  – Fun: Games with a Purpose (GWAP)

• Efficiency

• Quality Control:
  – Duplication
  – Sentinel Operations
  – Self-Refereeing
Algorithm 1: Depth Layers
Calculate Depth of a Given Image?
Calculate Depth of a Given Image?

• **Automatic methods:**
  – Depth increases in the up direction
  – Color similarity implies depth similarity
Calculate Depth of a Given Image?

• Automatic methods:
  – Depth increases in the up direction
  – Color similarity implies depth similarity

• Not always correct
Calculate Depth of a Given Image?

- Automatic methods:
  - Depth increases in the up direction
  - Color similarity implies depth similarity

- Not always correct
Calculate Depth of a Given Image?

- Automatic methods:
  - Depth increases in the up direction
  - Color similarity implies depth similarity

- Not always correct

- Some images are very challenging (art)
Micro Task?
Micro Task?

• Ask “what is the depth of the pixel?”
  – Too fine, can be ambiguous
Micro Task?

• Ask “what is the depth of the pixel?”
  – Too fine, can be ambiguous
• Ask “what is the depth of an object?”
  – Segmentation is too complex
Micro Task?

• Ask “what is the depth of the pixel?”
  – Too fine, can be ambiguous

• Ask “what is the depth of an object?”
  – Segmentation is too complex

• Ask “what is the depth of a patch in the image?”
  – Getting better... but humans are not good at assessing absolute depth
Relative Ordering

• Ask “which is closer” on neighboring patches?
Relative Ordering

• Ask “which is closer” on neighboring patches?
  – Reliable, but not well-defined. A is closer than B:
Our Micro Task

Is there a jump between the red region and the blue region, in terms of distance from the camera?

Place the mouse over an image to hide the highlighted regions.

○ No, there is no jump between the red and blue regions.
○ Yes, and the blue region is farther from the camera.
○ Yes, and the red region is farther from the camera.

Yes, and the blue region is farther from the camera.

Yes, and the red region is farther from the camera.

No, there is no jump between the red region and the blue region, in terms of distance from the camera.
Guidelines for Choosing Tasks

• Task must be simple (instantaneous)
• Task must be specific (well-defined)
• Task must be reliable (humans can do it)
Combining
Combining

- Laplace equation $\Delta f = 0$ with constraints
Algorithm

DEPTH-LAYERS(image I, sentinel queries S)

1. Segment I into regions (using mean-shift and SLIC)
2. Insert all pairs of neighboring regions into Q
3. loop in parallel until each pair has been visited \( N \) times
   - Gather \( K \) random pairs from \( Q \)
   - Gather \( M \) random pairs from \( S \)
4. for each pair: Build the visual query & Duplicate it
   - Mix the \( 2K + 2M \) queries
5. results = send all queries to an HP
6. if \( \text{average}(\text{consistent(results)}) \geq 0.75 \)
   - for each pair
     - Add consistent results to the list of votes
     - Increment #visited
7. for each pair of neighboring regions
   - final_result = majority(list of votes)
8. Solve the Laplace equation to construct a depth map
Algorithm

\textbf{DEPTH-LAYERS}(image }I\text{, sentinel queries }S\text{)

1. Segment image regions using mean-shift and SLIC
2. Insert all pairs of neighboring regions into }Q\text{ }
3. \textbf{loop in parallel until} each pair has been visited }N\text{ times
4. Gather }K\text{ random pairs from }Q\text{ }
5. Gather }M\text{ random pairs from }S\text{ }
6. \textbf{for} each pair: Build the visual query & Duplicate it
7. Mix the }2K + 2M\text{ queries
8. }results = \text{ send all queries to an HP
9. \textbf{if} }\text{average}(\text{consistent}(results)) \geq 0.75\text{ }
10. \textbf{for} each pair
11. \text{Add consistent results to the list of votes
12. Increment }\#\text{visited
13. \textbf{for} each pair of neighboring regions
14. }final\_result = \text{majority}(\text{list of votes)
15. Solve the Laplace equation to construct a depth map
Algorithm

DEPTH-LAYERS(image \(I\), sentinel queries \(S\))

1. Segment image regions using mean-shift and SLIC)
2. Insert all pairs of neighboring regions into \(Q\)
3. \textbf{loop in parallel until} each pair has been visited \(N\) times
4. Gather \(K\) random pairs from \(Q\)
5. Gather \(M\) random pairs from \(S\)
6. \textbf{for} each pair: Build the visual query & Duplicate it
7. Mix the \(2K + 2M\) queries
8. \(\text{results} = \text{send all queries to an HP}\)
9. \textbf{if} average(consistent(results)) \(\geq 0.75\)
10. \textbf{for} each pair
11. Add consistent results to the list of votes
12. Increment #visited
13. \textbf{for} each pair of neighboring regions
14. \(\text{final_result} = \text{majority}($\text{list of votes}$)
15. Solve the Laplace equation to construct a depth map
Algorithm

DEPTH-LAYERS(image I, sentinel queries S)

1. Segment image regions using mean-shift and SLIC
2. Insert all pairs of neighboring regions into Q
3. loop in parallel until each pair has been visited N times
   a. Gather K random pairs from Q
   b. Gather M random pairs from S
4. for each pair: Build the visual query & Duplicate it
5. Mix the 2K + 2M queries
6. results ← send all queries to an HP
7. if average(consistent(results)) ≥ 0.75
8.   for each pair
9.     Add consistent results to the list of votes
10.    Increment #visited
11. for each pair of neighboring regions
12.   final_result = majority(list of votes)
13. Solve the Laplace equation to construct a depth map
Algorithm

DEPTH-LAYERS(image I, sentinel queries S)
1  Segment using regions using mean-shift and SLIC)
2  Insert all pairs of neighboring regions into Q
3  loop in parallel until each pair has been visited N times
4   Gather K random pairs from Q
5   Gather M random pairs from S
6  for each pair: Build the visual query & Duplicate it
7   Mix the 2K + 2M queries
8   results = send all queries to an HP
9   if average(consistent(results)) ≥ 0.75
10  for each pair
11   Add consistent results to the list of votes
12   Increment #visited
13  for each pair of neighboring regions
14   final_result = majority(list of votes)
15  Solve the Laplace equation to construct a depth map
Algorithm

DEPTH LAYERS(image I, sentinel queries S)
1 Segment image regions using mean-shift and SLIC
2 Insert all pairs of neighboring regions into Q
3 loop in parallel until each pair has been visited N times
4 Gather K random pairs from Q
5 Gather M random pairs from S
6 for each pair: Build the visual query & Duplicate it
7 Mix the 2K + 2M queries
8 results ← send all queries to an HP
9 if average(consistent(results)) ≥ 0.75
10 for each pair
11 Add inconsistent results to the list of votes
12 Increment #visited
13 for each pair of neighboring regions
14 final_result ← majority(list of votes)
15 Solve the Laplace equation to construct a depth map
absolute depth

discrete depth

31
relative depth

discrete depth

relative depth
Algorithm 2: Normal Map

Orient the thumbtacks flush against the surface.

The thumbtack’s pin should point away from the surface behind it. See the Example for good and bad examples.

Thumbtacks may appear at the same location multiple times. We check for consistency and may reject inconsistent HITs.
Algorithm 3: Bilateral Symmetry Map

Move the green circle so it is symmetric to the yellow circle.

If the yellow circle is over a point on the left side of the body, place the green circle over the same point on the right side. See the Example for good and bad examples.

Dots may appear at the same location multiple times. We check for consistency and may reject inconsistent HITs.

The point symmetric to the yellow circle is not visible in the image.

Hide circles.
Some Statistics

<table>
<thead>
<tr>
<th>example</th>
<th>micro-tasks used</th>
<th>ratio of used per executed</th>
<th>$ per micro-task</th>
<th>total $ cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal map</td>
<td>1620–4340</td>
<td>0.60</td>
<td>.002–.003</td>
<td>$5.04–10.76</td>
</tr>
<tr>
<td>depth layers</td>
<td>2669–7620</td>
<td>0.76</td>
<td>.002</td>
<td>$6.41–17.15</td>
</tr>
<tr>
<td>symmetry map</td>
<td>1020–1740</td>
<td>0.93</td>
<td>.002</td>
<td>$3.24–3.92</td>
</tr>
</tbody>
</table>

Table 1: *Micro-tasks*

<table>
<thead>
<tr>
<th>example</th>
<th>total HPs</th>
<th>% completely unreliable</th>
<th>average reliability for reliable HPs</th>
<th>micro-tasks per HP avg median</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal map</td>
<td>61</td>
<td>42%</td>
<td>89%</td>
<td>123 33</td>
</tr>
<tr>
<td>depth layers</td>
<td>48</td>
<td>35%</td>
<td>87%</td>
<td>193 63</td>
</tr>
<tr>
<td>symmetry map</td>
<td>19</td>
<td>24%</td>
<td>99%</td>
<td>97 20</td>
</tr>
</tbody>
</table>

Table 2: *Human Processors*
## Timing

<table>
<thead>
<tr>
<th>example</th>
<th>successful micro-task duration</th>
<th>algorithm delay until</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>avg</td>
<td>median</td>
</tr>
<tr>
<td>normal map</td>
<td>8.8 s</td>
<td>8.1 s</td>
</tr>
<tr>
<td>depth layers</td>
<td>6.2 s</td>
<td>5.5 s</td>
</tr>
<tr>
<td>symmetry map</td>
<td>9.0 s</td>
<td>8.5 s</td>
</tr>
</tbody>
</table>
Accuracy
Accuracy
Accuracy

\begin{figure}
\centering
\includegraphics[width=\textwidth]{accuracy_plot}
\caption{Accuracy of the method for different number of HPs (N). The color scale represents the probability of accurately identifying a reliable HC based on the sentinel threshold.}
\end{figure}
Conclusions

• For hard problems, HC algorithms can beat automatic algorithms.
• Rephrase your problem in terms of reliable human perception.

• How can we improve efficiency?
• If this were a Photoshop plug-in, how much would people pay to use it?
End
Related Work (1/6)

- Many kinds of collective intelligence
  - open-source software, Wikipedia, PageRank, supervised learning, elections?
- Modern assembly line (Ford Motor Company 1908–1915)
- Interchangeable parts:
  - Adam Smith on division of labor (1776)
  - Terracotta army (3rd century BC)
  - Venetian Arsenal (ship building)
Related Work (2/6)

• Online:
  – [von Ahn 2008]
  – [Little et al. 2010a,b] and [Bernstein 2010]
  – [Bigham et al. 2010] and [Bernstein 2011]
  – [Davis et al. 2010]
  – [Sorokin et al. 2010]
  – many more recent/contemporary applications

• Recast existing experiments
  – [Koenderink et al. 1992], [Cole et al. 2009]
  – [Chen et al. 2009]
Related Work (3/6)

• Training data:
  – ESP Game [von Ahn and Dabbish 2004], ...
  – LabelMe [Russel et al. 2008; Yuen et al. 2009]
  – Hands by Hand [Spiro et al. 2010]

• Using HC data gathered offline:
  – [Talton et al. 2009]
Related Work (4/6)

• Depth Layer Algorithm
  – automatic: [Hoiem et al. 2005; Assa and Wolf 2007; Saxena et al. 2009]

• Normal Map Algorithm
  – manual: [Wu et al. 2008]

• Symmetry Map Algorithm
  – automatic: [Chen et al. 2007]
Related Work (5/6)

• History
  – Genetic Algorithms
    • [Sims 1991]
    • Interactive Genetic Algorithm [Takagi 2001]
    • Human-Based Genetic Algorithms [Kosorukoff 2001]
    • Electric Sheep
  – Open Mind Initiative
  – collaborative filtering: [Goldberg et al. 1992; Adomavicius and Tuzhilin 2005]
• “Human Computation” [von Ahn 2005]
Related Work (6/6)

• Recent survey: [Quinn and Bederson 2011]

• Market properties:
  – [Ipeirotis 2010; Chilton et al. 2010; Faridani et al. 2011; Mason and Suri 2011; Mason and Watts 2010]

• Surface perception:

• Shape-from-Shading:
  – [Durou et al. 2008]
Theoretical Limits

• 125–180 seconds (median) / 20 questions = 6.25–9 seconds per perception for our tasks
• 7 billion humans (does not include other animals capable of similar tasks)
• (number of humans) / (seconds per perception) ~ = 1 billion perceptions per second