Addressing Challenges with Augmented Reality Applications on Smartphones

Krzysztof Zienkiewicz

krzysztof.k.zienkiewicz@vanderbilt.edu

Vanderbilt University
Nashville, Tennessee

Presented at Mobilware 2010
Augmented Reality (AR)

- Azuma’s definition: AR is a system that has the following characteristics
  - Combines real and virtual
  - Interactive in real time
  - Registered in 3D

- Milgram’s Continuum
Components of an AR application

- Rendering 3D context-specific data
- Retrieving geotagged Points Of Interest (POIs)
- Frame of Reference estimation
Challenges

• Mobile 3D solutions are not optimal and hard to mesh with camera imagery
• Filtering geotagged POIs by proximity is computationally intensive
• Real-time estimation of frame of reference is computationally demanding
• Geomagnetic sensor noise makes orientation estimation hard
Mobile 3D solutions are not optimal

- AR needs precise 3D overlays to provide an intuitive interface
- Without accuracy, user experience deteriorates
- Meshing 3D data with camera preview is highly platform dependent
- Mobile GLs don’t offer a “pick” mode
Retrieving POIs by proximity is expensive

- Areas of interest may have a high density of POIs
- Retrieval by location requires many comparisons
- Bandwidth limitations
Estimation of frame of reference

- Fiducial markers
- Feature extraction
Geomagnetic sensor noise

- Compass used to estimate the field of vision
- Compass noise introduces jitter in the rendered overlays
- Savitzky-Golay smoothing filter is too slow
- Noise +/- 9 degrees
Geomagnetic sensor noise

Device Steady

Angle offset (degree)

Time (s)
Effects of geomagnetic sensor noise
Effects of geomagnetic sensor noise

Effect of Compass Noise

\[ \text{jitter} = \frac{\text{width}}{2 \tan \alpha} \tan \theta \]
Solutions

• Using hardware accelerated 3D APIs to display rendered content
• A grid-based approach to data storage and retrieval
• Using GPS and geomagnetic sensors to estimate device position and orientation
• Statistical analysis filter
Hardware accelerated 3D APIs

- Apple's UIKit allows a transformation matrix to be applied to views
  - Allows for hit testing

- Other platforms will need to do hit testing by hand
  - But get to use GLs for rendering

- Meshing achieved via layering

\[
\begin{bmatrix}
\frac{F}{\text{aspect}} & 0 & 0 & 0 \\
0 & F & 0 & 0 \\
0 & 0 & \frac{n+f}{n-f} & \frac{2fn}{n-f} \\
0 & 0 & -1 & 0
\end{bmatrix}
\]
Grid-based data storage and retrieval

- Mapping function takes location to grid indices
- Each square contains all POIs within the region
- No numeric comparisons are performed by the server
- POIs are downloaded in bulk and need to be filtered on the phone

<table>
<thead>
<tr>
<th>Query Type</th>
<th>Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latitude range (latitude column indexed)</td>
<td>581700 μs</td>
</tr>
<tr>
<td>Latitude range (latitude column not indexed)</td>
<td>284600 μs</td>
</tr>
<tr>
<td>Specific latitude bucket</td>
<td>5209 μs</td>
</tr>
</tbody>
</table>
Using sensors to estimate field of view

- The computationally cheap solution
- Only need to update the transformation matrix
Nature of the noise

Noise Normality Test

y = 1.0036x - 0.0075
R² = 0.9861
Filtering algorithm

Variables/Functions:
- $R =$ Ring Buffer of Received Data
- $O =$ Ring Buffer of Outlier Data
- $|R| = |O| =$ Maximum Allowable Size of Buffer
- size(buffer) = ReturnsCurrentSizeofBuffer
- $p_i =$ A compass reading as a Single Precision Float
- $Z(p_i) = (p_i - \text{mean}(R))/\text{stdDev}(R)$
- $Z_{range} =$ Maximum Allowable Deviation
- outlierDirection($p_i$) = $p_i > \text{mean}(R)?1:-1$
- enqueue(buffer,$p_i$) = Adds $p_i$ to the Buffer

Algorithm:
- filtered($p_i$) =
  - if $\text{size}(R) < |R|$: enqueue($R,p_i$)
  - else:
    - $z_i = Z(p_i)$
    - if $\text{abs}(z_i) \leq Z_{range}$:
      - enqueue($R,p_i$)
      - clear($O$)
    - else: enqueue($O,p_i$)
  - if $\text{size}(O) = |O|$:
    - side = outlierCluster()
    - $\forall p_j \in O$
      - if outlierDirection($p_j$) = side:
        - enqueue($R,p_j$)
        - clear($O$)
    - return $\text{mean}(R)$

outlierCluster() =
- int sum = 0
- $\forall p_j \in O$
  - sum += $p_j - \text{mean}(R)$
- return signum(sum)

- Small number of parameters
- Extendable for dynamic parameterization
- Good results: 60% noise reduction
Filtering algorithm

Device Steady
Filtering algorithm

Device during Uniform Rotation

Angle offset (degree)

Time (s)
Thanks

Questions?