Client-side backprojection of presentation slides into educational video

Yekaterina Kharitonova†*, Qiyan Tung†, Alexander Danehy†, Alon Efrat†, Kobus Barnard†‡*

University of Arizona: †Department of Computer Science, ‡School of Information: Science, Technology and Arts; *The iPlant Collaborative

Introduction

Motivation:
The slide area in blurry, low-quality or compressed videos is hard to read. Displaying clear and sharp slides is more informative than showing the speaker, background, and audience with the same level of detail.

Idea:
Blacking-out the slide area in the video results in smaller-sized video files, thus, reducing bandwidth. HTML5 technology allows to reconstruct the video using slide-to-frame homographies.

Goal:
Backproject high-resolution slide images into the video stream on the client side.

Backprojection

We transform homogeneous slide points, \( s = [x, y, w]^T \), into the frame coordinates, \( p = [u, v, w]^T \), by applying a homography \( H \) : \( p = H s \).

We can approximate \( H \) using an affine transformation: the camera events such as zooming-in/-out, and panning may be approximated by scaling and translating the first frame in the event sequence.

Given the two consecutive frames \( F_0 \) and \( F_2 \), we can approximate points from \( F_0 \) by using a homogeneous matrix \( T \), instead of the full homography. \( T \).

Homography approximation

Let \( q^b \) be the image points on frame \( F_0 \), approximated by applying the affine transformation matrix \( T \) to the coordinates from the previous frame, \( p^a \): \( p^a = q^b = TP^a \).

To compute the homogeneous matrix \( T \):
1. Rewrite the matrix \( T \) as a column vector \( t = \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} \).
2. Place the coordinates of \( p^a \) into a matrix \( u \).
3. Arrange \( q^b \) as an 8x1 column vector, where \( q^b = Ut \).
4. Use the linear least squares to solve for \( t \) by minimizing the squared error, \( E = ||e||^2 \), where \( e = Ut - p^a \rightarrow t = (U^T U)^{-1} U^T e \).

System implementation

Define: \( r \) as the threshold of the average re-projection error (in pixels).

For each frame:
1. Compute the coordinates of the slide corners in the frame, \( p^a \).
2. Estimate the affine transformation \( (T) \) and the homography \( (H) \) between the successive frames.
3. Backproject the corners using \( T \) to get the approximated coordinates, \( q^a \).
4. If (\( q - p^a \) – \( r \)
   Then, use an affine transformation \( T \) for that frame.
   Else, use that frame’s homography \( H \).

True backprojections vs. homography approximation

If the client has sufficient CPU resources

Use: the direct homography computation

Bandwidth requirements: modified video, the slide images, timing and homography data.

If the client cannot run computationally-intensive operations

Use: the affine approximation

Bandwidth requirements: modified video, the transformed slide images, timing data.

Table: Data

<table>
<thead>
<tr>
<th>ID</th>
<th># of Slides</th>
<th>Slides/Video size</th>
<th># of Frames</th>
<th>Video size</th>
</tr>
</thead>
<tbody>
<tr>
<td>WC</td>
<td>48</td>
<td>1.7 Mb</td>
<td>71387</td>
<td>187.6 Mb</td>
</tr>
<tr>
<td>FB</td>
<td>37</td>
<td>1.2 Mb</td>
<td>87784</td>
<td>229.4 Mb</td>
</tr>
<tr>
<td>LA</td>
<td>103</td>
<td>3.6 Mb</td>
<td>91770</td>
<td>240.5 Mb</td>
</tr>
</tbody>
</table>

Presentation data statistics for the three videos. The size of the high-resolution slide deck is small compared to that of the video.

Video ID | HC | FB | LA
---|---|---|---
Compression Setting (kbps) | 600 | 400 | 200 | 600 | 400 | 200 | 600 | 400 | 200
Backprojected Video (Mb) | 171.9 | 129.1 | 52.7 | 227.8 | 159.4 | 90.4 | 227.6 | 165.2 | 95.2
Modified Video (Mb) | 102.9 | 75.3 | 47.1 | 135.4 | 107.6 | 73.5 | 117.8 | 96.1 | 70.3
Video Saving | 40.1% | 41.8% | 10.6% | 40.6% | 32.5% | 18.7% | 48.2% | 43% | 26.2%
Overall Saving | 39.1% | 40.5% | 7.4% | 40.0% | 31.7% | 24.7% | 46.5% | 40.9% | 22.4%

The sizes (in Mb) of the three backprojected videos and the modified videos, which were created by backprojecting a black slide. The overall savings take into account the slide image data.

HTML5 <canvas> element is a drawable bitmap region, which can be used to draw and script graphics using JavaScript. Its built-in transform() method only supports scale, translation and rotation. It also provides a native integration with the HTML <video> tag.

With canvas, we can create a JavaScript routine to manipulate a backprojected slide image independently from the video frame. This lets us overlay slide images over video frames directly in the client’s browser.

Additional Information

Visit http://slic.arizona.edu to learn more about the SLIC project and to watch the demo.