Scalable View-Dependent Progressive Mesh Streaming

WEI TSANG OOI
National University of Singapore
joint work with
Cheng Wei
National University of Singapore
Ammonite (Meekoceras gracilitatus)

Grab the object above with your mouse, using the combinations below.

<table>
<thead>
<tr>
<th>Rotate</th>
<th>Zoom</th>
<th>Translate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rotate</th>
<th>Zoom</th>
<th>Translate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hoppe’s Progressive Mesh

Edge Collapse

Vertex Split
At the sender

\[ V_k + \ldots + V_4 + V_3 + V_2 + V_1 = \text{base model} \]
Transmission

TCP

base model

V₁  V₂  V₃  V₄  \ldots  Vₖ

UDP
At the receiver

base model

$v_1$ $v_2$ $v_3$ $v_4$ ...

$v_k$
base mesh
complete mesh
view-dependent streaming: only send what the receiver can see
Relation between PSNR and received bytes

PSNR

Received Bytes (KB)

View-dependent

View-independent
what to send?

in what order?
what to send?
* determined by viewpoint *

in what order?
* determined by visual contributions *
Existing Approach

- View point
- What to split
- How to split
Existing Approach

view point

what to split

how to split
For each receiver, server needs to:

- compute visibility
- compute visual contribution of each vertex split
- sort vertex splits
- remember what has been sent
“dumb client, smart server”
does not scale
Receiver-driven Approach
how to identify a vertex split?
Attempt 1
want to split vertex 2

here is how to split, and 2 splits into 6 and 7
Kim, Lee, “Truly selective refinement of progressive meshes,”
want to split vertex 00
here is how to split 00
Receiver-driven Approach

- what to split
- how to split
Encoding of vertex split IDs
proc encode(T)

if no vertices to be split in T
    return 0

else
    return 1 + encode(T.left) + encode(T.right)
Encoding of vertex split IDs

0 100 1000 1 100 1 1000
how to compute visibility + visual contributions?
(without possessing the complete mesh?)
Estimate with screen space area of vertices
<table>
<thead>
<tr>
<th></th>
<th>Sender-driven</th>
<th>Receiver-driven</th>
</tr>
</thead>
<tbody>
<tr>
<td>send base mesh</td>
<td>1.4</td>
<td>1.13</td>
</tr>
<tr>
<td>decode IDs</td>
<td>-</td>
<td>1.55</td>
</tr>
<tr>
<td>search vertex split</td>
<td>1.85</td>
<td>1.85</td>
</tr>
<tr>
<td>determine visibility</td>
<td>0.41</td>
<td>-</td>
</tr>
<tr>
<td>update state</td>
<td>1.41</td>
<td>-</td>
</tr>
<tr>
<td>encode IDs</td>
<td>0.94</td>
<td>-</td>
</tr>
<tr>
<td>others</td>
<td>0.16</td>
<td>0.16</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>6.17</strong></td>
<td><strong>4.69</strong></td>
</tr>
</tbody>
</table>
receiver-driven protocol alleviates the computational bottleneck at the sender.
the other bottleneck is bandwidth.
**goal**: reduce server overhead by retrieving vertex splits from other clients if possible
**difficulty:** need to **quickly** and **efficiently** determine who to retrieve the vertex splits from
low server overhead

low response time

low message overhead
common P2P techniques:

1. build an overlay and push
2. use DHT to search for chunks
3. pull based on chunk availability
common P2P techniques:

1. build an overlay and push

2. use DHT to search for chunks

3. pull based on chunk availability
peer-to-peer file transfer:

*a needed chunk is likely to be available in any peer*
peer-to-peer video streaming:

a needed chunk is likely available from a peer that has watched the same segment earlier

(temporal locality)
peer-to-peer mesh streaming

a needed chunk is likely available from a peer that is viewing the same region

(spatial locality)
idea: exploit spatial locality to reduce message overhead.
chunks
chunks

(1 chunk = 240 vertex splits)
groups

(1 group = 16 chunks)
Only exchange messages between peers that need chunks from the same group.
how the protocol works
server maintains a list of group members for each group, and who possesses which chunk.

(128.3.13.44, 100100) (123.44.121.99, 111111) ..
(90.1.1.00, 0001001) (32.11.99.233, 101111) ..

..
client: “I want to view mesh M”

server sends:
(i) base mesh
(ii) group members of the highest group.
(iii) what each member possess
client decides which vertex splits (chunk) to refine

if some peer has that chunk, request from peer else request chunk from server
peers inform server when they received a chunk

if a chunk in the next group can be decoded, server sends group members of the next group
if too many group members, server sends only most recent subsets + some seeds
on-going work:

1. evaluation using user traces and simulator

2. other design parameters

3. further reduce the role of server
summary

receiver-driven design to reduce CPU cost

peer-to-peer design to reduce bandwidth cost
謝謝