MESH ENCODINGS FOR X3DOM: RECENT ADVANCES

Max Limper
Fraunhofer IGD
Fraunhoferstraße 5
64283 Darmstadt, Germany
Tel.: +49 (0) 6151 155 – 554
Fax.: +49 (0) 6151 155 – 105
Email: max.limper@igd.fraunhofer.de
http://www.igd.fraunhofer.de/vcst
Outline

- Introduction
  - Declarative 3D Scalability Issues
  - Large Model Visualization with X3DOM
  - Declarative 3D Mesh Encoder: Key Aspects

- X3DOM Mesh Encodings
  - Sequential Image Geometry
  - Progressive Binary Encoder
  - POP (Progressively ordered Primitive) Buffer
Motivation:
X3DOM / Declarative 3D Scalability Issues

- **X3DOM / Declarative 3D**
  - Simplifies 3D WebApp development by extending HTML/DOM with new 3D elements

- **Scalability issues**: mesh/anim data, encoded as text, leads to huge pages

- **Solution**: external asset container, requested on demand. (Similar to `<img>`)
Motivation:
Large Model visualization with X3DOM

- Out-Of-Browser Based Rendering (Out-Of-Core with W3C Technology)
  - **Goal**: Great user experience with massive datasets
  - **Solution**: Hybrid tree structure for client and server based visibility calculation which provides application controlled priorities

- Core Challenges:
  - How to efficiently page mesh data?
  - How to manage loading on demand?
Declarative 3D Mesh Encoder

Key Aspects

- Compact Representation
- Web3D Format
- Fast Decompression
- Progressive Transmission
Declarative 3D Mesh Encoder

Contradictory Constraints

- **Compact Representation**
  - Externalize binary mesh data from HTML documents (*TypedArray* Spec.)
  - Integer quantization of floating-point mesh attributes (e.g., 16 bit)
  - Compression algorithms?

- **Progressive Transmission**
  - Progressive meshes (Hoppe) difficult for generalized 3D data
  - Progressive transmission ↔ fast decoding, fast GPU upload?

- **Fast Decompression**
  - Ideal: Download mesh data, direct GPU upload
  - Still efficient: Exploit common HTTP / browser features (Images, GZIP, …)
Sequential Image Geometry

- 2010: No support for binary transmission with XHR/W3C
- **Solution:** Utilize images for meshes
  - Browser/Server are well-optimized for parallel fetching/decoding
  - Images can directly be uploaded to GPU buffers (textures)

Sequential Image Geometry

- Implicit mesh does not correlate with the mesh topology

- Application
  - Supports: Transmission, compression (partially), rendering, ...
  - Does not support: Scaling of SIG container
  - Works with any mesh type and keeps the original topology
Sequential Image Geometry
Bytes per triangle

- Bunny X3DB
- Bunny SIG-Raw 16bit
- Bunny SIG-PNG 16bit
- Horse X3DB
- Horse SIG-Raw 16bit
- Horse SIG-PNG 16bit
Sequential Image Geometry
Image Streaming and compression

- Encode mesh animation in video encoder/decoder

- Optimize compression by resorting mesh components

Coordinate Image (Detail):
91.4 KB → 44.7 KB
Progressive Binary Container

- 2012: TypedArray (Binary Buffer) support in JavaScript / XHR (W3C)
- Utilize TypedArray container for meshes
  - TypedArray container can directly be uploaded to GPU buffer
  - Gzip compression supported by the browser
- Progressive Updates similar to SIG, but not limited through fixed-size (e.g., 8 bit) image channels

Progressive Binary Container

- **Advantage**
  - Simple progressive updates by refining the quantization level

- **Disadvantages**
  - Transmits always the full topology / number of triangles → no LOD
  - Costly GPU updates (bit shifts and adds)
Binary Mesh Container Comparison

**Case Study:** Compact Representation ↔ Fast Decompression

Desktop

<table>
<thead>
<tr>
<th>Bandwidth (kbit/s)</th>
<th>Startup Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>5000</td>
</tr>
<tr>
<td>4000</td>
<td>500</td>
</tr>
<tr>
<td>16000</td>
<td>50</td>
</tr>
<tr>
<td>50000</td>
<td>5</td>
</tr>
</tbody>
</table>

iPad 3

<table>
<thead>
<tr>
<th>Bandwidth (kbit/s)</th>
<th>Startup Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>5000</td>
</tr>
<tr>
<td>4000</td>
<td>500</td>
</tr>
<tr>
<td>16000</td>
<td>50</td>
</tr>
<tr>
<td>50000</td>
<td>5</td>
</tr>
</tbody>
</table>

**Still missing:** Progressive topology refinement and LOD

X3DOM POP Buffer: Simplified progressive transmission

- Progressive Meshes? (Hoppe ‘96, Alliez/Desbrun ‘01, Peng ‘05, …)
  - Most don’t encode general 3D meshes, just 2-manifolds
  - Tradeoff: Bandwidth Usage vs. Decode Time
  - Web & Mobile clients: Limited computational power

- Simple Web-capable Implementation, as known from Images?
Progressive Transmission: Interlacing Schemes

Images (PNG, Adam7)

Meshes (POP Buffer)

http://beej.us/blog/data/image-interlacing/

Progressively Ordered Primitive (POP) Buffer

- Coarse **quantization** of geometry (e.g., 5 Bits) → **degenerate triangles**
- Successive removal of degenerate triangles at each quantization level

- **Reordering** of triangle data according to degeneration property

- Progressive **streaming** and **LOD**
X3DOM POP Buffer
Progressive Streaming and LOD

(a) 4 bit, 3% loaded
(b) 5 bit, 13% loaded
(c) 6 bit, 38% loaded

Graphs showing the progression of triangles and vertices with increasing precision levels.
X3DOM POP Buffer

Benefits

- Handles arbitrary topology
- Conversion at interactive rates
- No CPU-based decoding steps
- Stateless structure (less GPU traffic, Instancing)
- No GPU memory overhead
- Straightforward implementation (WebGL)
Try it yourself

- Paper video and demos available at www.x3dom.org/pop

- Demos need WebGL-capable browser
Thank you for listening!

Questions?
Appendix:
POP Buffer Details
Ordering Triangles: Finding degenerated geometry

Non-degenerate triangles:

\[ Q_2 = \{B, C, D, H, I\} \]
\[ Q_3 = \{A, B, C, D, E, F, G, H, I\} \]
Ordering Triangles: Sorting

- Classify triangles, sort them
- Order within sets can be freely chosen
- Fast sorting (linear time), ~5-6 Mio. Δ/s
Progressively Ordered Primitive (POP) Buffer
Progressive Transmission & Rendering

- Don’t send less bits/vertex, just send less vertices!
  - Also applies for index data, normals, texture coordinates
  - No additional decoding, guarantees **stateless** property
The Stateless POP Buffer

- One static buffer for all detail levels
- One static buffer for all instances
View-Dependent LOD: Closing Cracks

- Partition mesh for view-dependent LOD
- Problem: Cracks between sub-meshes
- Simple solution: Protected vertices
  - Move border vertices to begin of buffer
  - Always render those with full precision
  - Can be used for shape / feature preservation