Rendering of Stylized Castle
WebGL demo
Oleksandr Popov

November 7, 2023
Stylized Castle WebGL demo

- Demo has stylized art direction - mostly untextured
- Sources - https://github.com/keaukraine/webgl-stylized-castle
- Implemented with a low-level custom WebGL framework
  - Framework is identical to Java Android framework
  - There is the same Android live wallpaper app
Scene composition

- Kitbashed static geometry
- Simple shadow maps
- Animated objects
  - Knights
  - Eagles
  - Flags
- Wind stripes
Static geometry

Models are exported from Asset Forge editor as batched geometry. Per-vertex data is 12 bytes:

- Position: 3 FP16
- Normal: signed byte
- Color index: unsigned byte

<table>
<thead>
<tr>
<th>Position</th>
<th>Normal</th>
<th>Col</th>
<th>padding</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>i</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>
Static geometry

Considered using packed \texttt{GL\_INT\_2\_10\_10\_10\_REV} for positions
This doesn’t provide enough precision - roughly 1 m per 1 km
Shadow maps

- Resolution: 2048x2048 px
- Hardware bilinear filtering
- 5-tap PCF
Shadow maps - PCF

PCF stands for percentage closer filtering. Basically this is sampling of nearby texels and calculating the average color of them.
Shadow maps - PCF

Unfiltered
Shadow maps - PCF

5-tap PCF unfiltered
Shadow maps - PCF

9-tap PCF unfiltered
Shadow maps - PCF

Filtered
Shadow maps - PCF

5-tap filtered
Shadow maps - PCF

9-tap filtered
Shadow maps - PCF

5-tap filtered
Shadow maps - acne

To avoid shadow acne two biases are applied:

- In vertex shader, a small bias is applied to offset geometry
- In fragment shader, a bias of 1.5 texels is applied before sampling shadowmap
- Both values are hand-picked and adjustable
Shadow maps - Performance

Update shadows at half frame rate.

- Often used in console games
- Virtually unnoticeable by end users
- Especially with high-refresh rate (90+ Hz)
Shadow maps - Performance

Use PCF only for nearby fragments.

- Introduces branching to fragment shader
- But reduces memory bandwidth
- Approximately 25% less texture units utilization on Pixel 7a
Shadow maps - depth-only shaders

- Used to render to depth map with light camera
- Don’t need to provide color output
- Have empty fragment shaders
Shadow maps - depth-only shaders

- Used to render to depth map with light camera
- Don’t need to provide color output
- Have empty fragment shaders

```
#version 300 es
precision mediump float;

void main(void) {
}
```
Shadow maps - depth-only shaders

- Used to render to depth map with light camera
- Don’t need to provide color output
- Have empty fragment shaders

```glsl
#version 300 es
precision mediump float;

void main(void) {
}
```
Shadow maps - depth-only shaders

Typical shader output:

- **Color** (what you see on screen)
- **Depth** (written to Z-buffer to cull fragments)
Shadow maps - depth-only shaders

Depth-only shader output:

No color
(nothing to see on screen)

Depth
(shadow map)
Shadow maps - depth-only shaders

Final scene without and with shadows:
Animations

Animated objects:

- Knights
- Birds
- Flags
- Wind stripes

No baked skeletal or vertex animations are used.

All animations are procedural - performed in vertex shader.
Animations - knights
Animations - knights

- Vertex data has only positions, normals and texture coordinates.
- Positions and UVs of different body parts can overlap.
- Model is “split” into different parts by grouping vertex data.
- Animation is controlled by 3 rotations passed via uniforms.

<table>
<thead>
<tr>
<th>Vertex indices</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-35</td>
</tr>
<tr>
<td>36-71</td>
</tr>
<tr>
<td>72-107</td>
</tr>
<tr>
<td>108-143</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Body</th>
<th>Head</th>
<th>Left arm</th>
<th>Right arm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-35</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36-71</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72-107</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>108-143</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Animations - knights

Vertex shader code:

```cpp
if (gl_VertexID < 36) { // body
}
else if (gl_VertexID < 72) { // head
    mat4 matHeadRotation = rotationAroundZ(headRotationZ);
    normal *= matHeadRotation;
    vertex *= matHeadRotation;
} else if (gl_VertexID < 108) { // left arm
    mat4 matLeftArmRotation = rotationAroundY(armRotations.x);
    normal *= matLeftArmRotation;
    vertex.z += ARM_PIVOT;
    vertex *= matLeftArmRotation;
    vertex.z -= ARM_PIVOT;
} else { // right arm
    mat4 matRightArmRotation = rotationAroundY(armRotations.y);
    normal *= matRightArmRotation;
    vertex.z += ARM_PIVOT;
    vertex *= matRightArmRotation;
    vertex.z -= ARM_PIVOT;
}
```
Animations - knights

- Models don’t have separate legs
- Bobbing is applied so knights walk, don’t slide
- Bobbing is an absolute value of sine wave
Animations - eagles

- Only wings are animated
- Single rotation is passed via uniform
Flags

- Inspired by wavy fish animation in ABZU
- Sine wave
- Reducing amplitude closer to origin (where it is attached to flagpole)
Wind stripes

• Inspired by wind effect in Sea of Thieves
• World-space random sine wave distortions
• Position offset to move
• Fade in and out
• Geometry is generated
• No buffers with vertex data are used
Wind stripes

Example: Render stripe of 20 segments
To do this, all is needed is to issue `glDrawArrays()` call with 20 * 3 * 2 triangles.
  • No buffers with vertex data are bound
  • Vertex shader doesn’t even have attributes
Wind stripes

Example: Render stripe of 20 segments
To do this, all is needed is to issue `glDrawArrays()` call with \( 20 \times 3 \times 2 \) triangles.

- No buffers with vertex data are bound
- Vertex shader doesn’t even have attributes
Wind stripes

VS creates segments based on $gl\_VertexID$
(test coloring is applied to each segment)
Wind stripes

Ends are tapered with \texttt{smoothstep}()
Wind stripes

XY distortion
Wind stripes

Z distortion
Wind stripes

Final result without test coloring of segments
Putting it all together

Ground+fog
Putting it all together

Static geometry
Putting it all together

Static geometry
Putting it all together

Flags
Putting it all together

Birds
Putting it all together

Knights
Putting it all together

Final scene with shadows
Thank you for watching!

Links to materials used in this presentation:

- Source code - https://github.com/keaukraine/webgl-stylized-castle
- Article with additional and more detailed explanations of rendering - https://keaukraine.medium.com/stylized-castle-webgl-demo-94aac2fde250
- Original 3D assets and Asset Forge editor used to create scene are by Kenney - https://www.kenney.nl/
A recording of this presentation will be available at
https://www.khronos.org/events/webgl-webgpu-meetup-november-2023

For more information on WebGL, please visit
https://www.khronos.org/webgl

Email: public_webgl@khronos.org