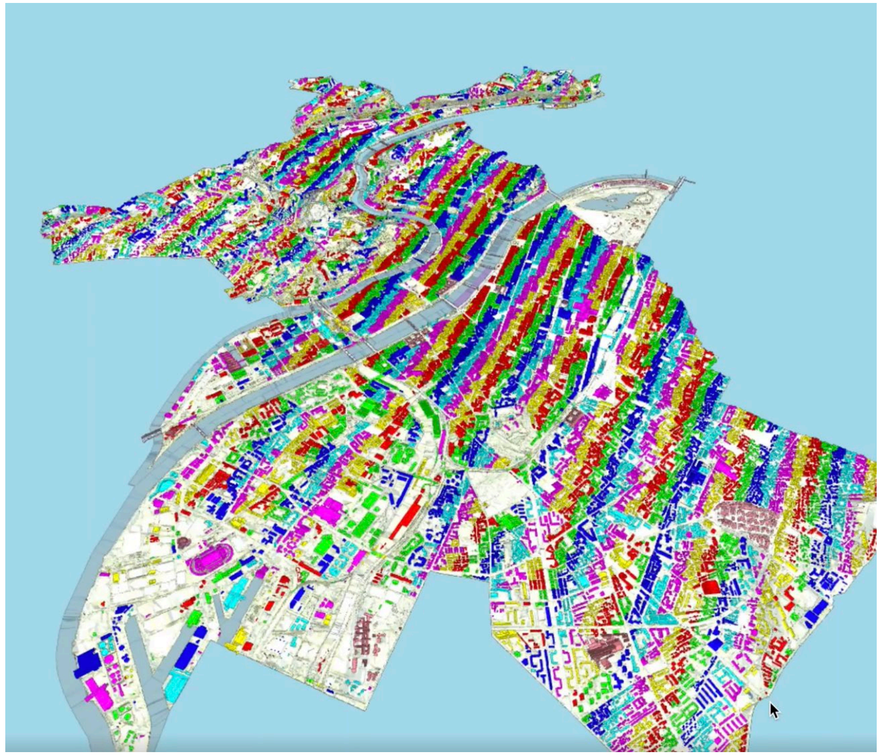
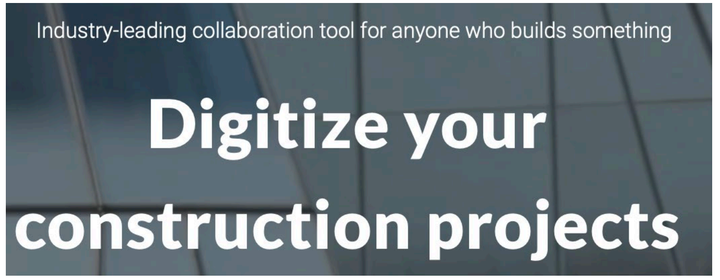
Creative RAM savings in WebGL

A tale about having fun with textures, and GLSL… in xeokit-sdk!



IMPORTANT!

Effort supported by:
www.tribia.com/en/



(part of) ***www.addnodegroup.com/en/home-page***

What is this about?

What is this about?

A way to aggressively reduce GPU-RAM consumption in WebGL:

What is this about?

A way to aggressively reduce GPU-RAM consumption in WebGL:

Using data textures…

What is this about?

A way to aggressively reduce GPU-RAM consumption in WebGL:

Using data textures…

- that contain “data”

What is this about?

A way to aggressively reduce GPU-RAM consumption in WebGL:

Using data textures…

- that contain “data” (indices, colors, ~~normals~~, matrices)

What is this about?

A way to aggressively reduce GPU-RAM consumption in WebGL:

Using data textures…

- *that contain “data” (indices, colors, ~~normals~~, matrices)*

- *read, **normalized** “data”!*

What is this about?

A way to aggressively reduce GPU-RAM consumption in WebGL:

Using data textures…

- *that contain “data” (indices, colors, ~~normals~~, matrices)*

- *read, **normalized** “data”!*

NUMBERS (GPU bytes/tri)

original xeokit-sdk: ~65 bytes/tri
new usage: 8~12 bytes/tri

Key ideas!

“Normalize data” => avoid duplicate data storage

Key ideas!

“Normalize data” => avoid duplicate data storage

unique positions - no normals - per object info

Key ideas!

“Normalize data” => avoid duplicate data storage
unique positions - no normals - per-object info

“Demoted” data-types => why 16 or 32 bit indices arrays?

Key ideas!

“Normalize data” => avoid duplicate data storage
unique positions - no normals - per-object info

“Demoted” data-types => why 16 or 32 bit indices arrays?
bucketting (8 bits if possible!) - re-bucketting (32 => 16 // 16 => 8)

Key ideas!

“Normalize data” => avoid duplicate data storage
unique positions - no normals - per-object info

“Demoted” data-types => why 16 or 32 bit indices arrays?
bucketting (8 bits if possible!) - re-bucketting (32 => 16 // 16 => 8)

One “gl.drawArrays” to rule them all!

Key ideas!

“Normalize data” => avoid duplicate data storage
unique positions - no normals - per-object info

“Demoted” data-types => why 16 or 32 bit indices arrays?
rucketting (8 bits if possible!) - re-bucketting (32 => 16 // 16 => 8)

One “gl.drawArrays” to rule them all!
for multiple batched objects - also N instances of M different obejcts

Key ideas!

“Normalize data” => avoid duplicate data storage
unique positions - no normals - per-object info

“Demoted” data-types => why 16 or 32 bit indices arrays?
rucketting (8 bits if possible!) - re-bucketting (32 => 16 // 16 => 8)

One “gl.drawArrays” to rule them all!
for multiple batched objects - also N instances of M different obejcts
bye bye gl.drawInstanced => from 2 fps to 8 fps*

How!?

How!?

PLACEHOLDER for boring comparison between C/C++ and GLSL

How!?

PLACEHOLDER for boring comparison between C/C++ and GLSL

Data-textures can contain “data” (positions / indices / colors)

How!?

PLACEHOLDER for boring comparison between C/C++ and GLSL

Data-textures can contain “data” (positions / indices / colors)

But can also contain metadata about the “data”!

- *object id’s, internal variables, … => “pointers”*

How!?

PLACEHOLDER for boring comparison between C/C++ and GLSL

Data-textures can contain “data” (positions / indices / colors)

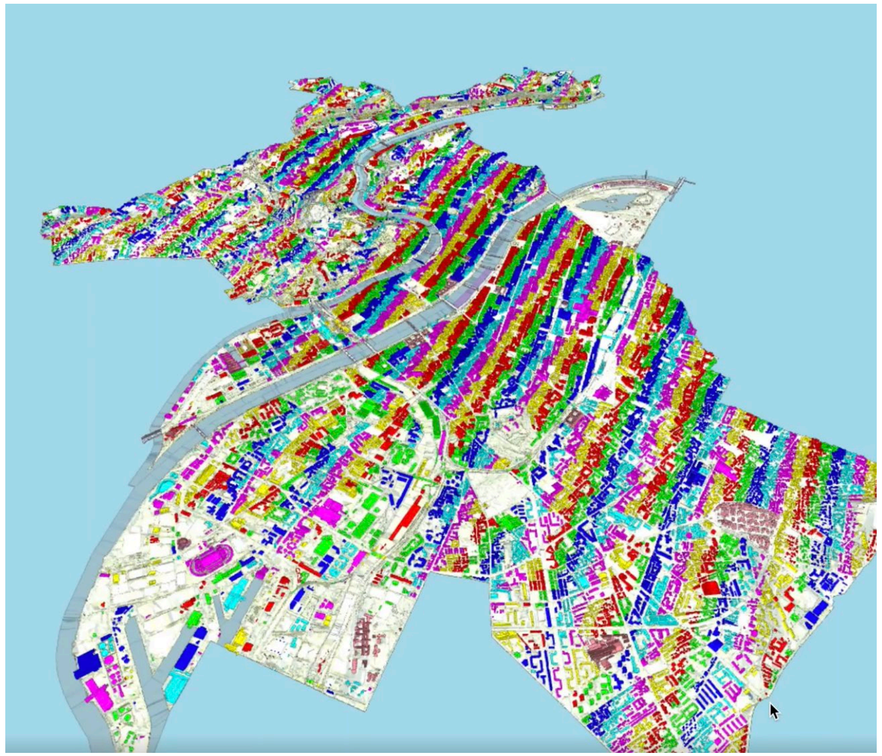
But can also contain metadata about the “data”!

- *object id’s, internal variables, … => “pointers”*

- *everything you can process “locally” (as in block compression)*
- ***goal: minimize data accesses***

Small demo… (see videos in link)

https://github.com/xeokit/xeokit-sdk/pull/824



… and a bit of code!

```
// constants
src.push("int polygonIndex = gl_VertexID / 3;")

// get packed object-id
src.push("int h_packed_object_id_index = (polygonIndex >> 3) & 1023;")
src.push("int v_packed_object_id_index = (polygonIndex >> 3) >> 10;")

src.push("int objectIndex = int(texelFetch(uTexturePerPolygonIdPortionIds, ivec2(h_packed_object_id_index, v_packed_object_id_index), 0).r)

// get flags & flags2
src.push("uvec4 flags = texelFetch (uTexturePerObjectIdColorsAndFlags, ivec2(2, objectIndex), 0);"); // chipmunk
src.push("uvec4 flags2 = texelFetch (uTexturePerObjectIdColorsAndFlags, ivec2(3, objectIndex), 0);"); // chipmunk

// flags.x = NOT_RENDERED | COLOR_OPAQUE | COLOR_TRANSPARENT
// renderPass = COLOR_OPAQUE

src.push(`if (int(flags.x) != renderPass) {`);
src.push("   gl_Position = vec4(3.0, 3.0, 3.0, 1.0);"); // Cull vertex
src.push("   return;"); // Cull vertex
src.push("} else {");

// get vertex base
src.push("ivec4 packedVertexBase = ivec4(texelFetch (uTexturePerObjectIdColorsAndFlags, ivec2(4, objectIndex), 0));");

src.push("ivec4 packedIndexBaseOffset = ivec4(texelFetch (uTexturePerObjectIdColorsAndFlags, ivec2(5, objectIndex), 0));");

src.push("int indexBaseOffset = (packedIndexBaseOffset.r << 24) + (packedIndexBaseOffset.g << 16) + (packedIndexBaseOffset.b << 8) + packed

src.push("int h_index = (polygonIndex - indexBaseOffset) & 1023;")
src.push("int v_index = (polygonIndex - indexBaseOffset) >> 10;")

src.push("ivec3 vertexIndices = ivec3(texelFetch(uTexturePerPolygonIdIndices, ivec2(h_index, v_index), 0));");
src.push("ivec3 uniqueVertexIndexes = vertexIndices + (packedVertexBase.r << 24) + (packedVertexBase.g << 16) + (packedVertexBase.b << 8) + packedVertexBase

src.push("ivec3 indexPositionH = uniqueVertexIndexes & 1023;")
src.push("ivec3 indexPositionV = uniqueVertexIndexes >> 10;")

src.push("mat4 positionsDecodeMatrix = mat4 (texelFetch (uTexturePerObjectIdPositionsDecodeMatrix, ivec2(0, objectIndex), 0), texelFetch (uTexturePerObjectI
src.push("mat4 entityMatrix = mat4 (texelFetch (uTexturePerObjectIdPositionsDecodeMatrix, ivec2(4, objectIndex), 0), texelFetch (uTexturePerObjectIdPosition
```

```
// get position
src.push("positions[0] = vec3(texelFetch(uTexturePerVertexIdCoordinates, ivec2(indexPositionH.r, indexPositionV.r), 0));")
src.push("positions[1] = vec3(texelFetch(uTexturePerVertexIdCoordinates, ivec2(indexPositionH.g, indexPositionV.g), 0));")
src.push("positions[2] = vec3(texelFetch(uTexturePerVertexIdCoordinates, ivec2(indexPositionH.b, indexPositionV.b), 0));")

// get color
src.push("uvec4 color = texelFetch (uTexturePerObjectIdColorsAndFlags, ivec2(0, objectIndex), 0);"); // chipmunk

src.push(`if (color.a == 0u) {`);
src.push("   gl_Position = vec4(3.0, 3.0, 3.0, 1.0);"); // Cull vertex
src.push("   return;");
src.push("};");

// get normal
src.push("vec3 normal = -normalize(cross(positions[2] - positions[0], positions[1] - positions[0]));");

src.push("vec3 position = positions[gl_VertexID % 3];");

src.push("vec4 worldPosition = worldMatrix * (positionsDecodeMatrix * vec4(position, 1.0)); ");

// get XYZ offset
src.push("vec3 offset = texelFetch (uTexturePerObjectIdOffsets, ivec2(0, objectIndex), 0).rgb;");

src.push("worldPosition.xyz = worldPosition.xyz + offset;");

src.push("vec4 viewPosition = viewMatrix * worldPosition; ");

src.push("mat4 entityNormalMatrix = mat4 (texelFetch (uTexturePerObjectIdPositionsDecodeMatrix, ivec2(8, objectIndex), 0), texe

src.push("vec4 worldNormal = entityNormalMatrix * worldNormalMatrix * vec4(normal, 1); ");

src.push("vec3 viewNormal = normalize((viewNormalMatrix * worldNormal).xyz);");
```