Khronos

glTF Sample Viewer

Moritz Becher

Lead GPU Software Engineer at UX3D GmbH

@UX3DGpuSoftware
ux3d.io
Native glTF 2.0 Viewer

WebGL 2.0 & JS ES6

Owned by Khronos
Focus:

- glTF spec conformity
- Simplicity
- Independence
- Functional **Sample Code**
Use Cases:

Verification of glTF implementation

Sample code

Quick & Easy glTF Preview
Metallic-Roughness Material

All parameters related to the metallic-roughness material model are defined under the `pbrMetallicRoughness` property of the `material` object. The following example shows how a material can be defined:

```gltf
vec3 BRDF_lambertian(vec3 f0, vec3 f90, vec3 diffuseColor,
{ 1.0 - specularWeight * F_Schlick(f0, f90, VdotH)}

// See https://seblagarde.wordpress.com/2012/01/08/pbr-materials/
vec3 BRDF_specularGGX(vec3 f0, vec3 f90, float alphaRoughness,
{ vec3 F = F_Schlick(f0, f90, VdotH);
float Vis = V_GGX(NdotL, NdotV, alphaRoughness);
float D = D_GGX(NdotH, alphaRoughness);
return specularWeight * F * Vis * D;
}

// f sheen
vec3 BRDF_specularSheen(vec3 sheenColor, float sheenRoughness,
{ float sheenDistribution = D_Charlie(sheenRoughness, NdotL);
float sheenVisibility = V_Sheen(NdotL, NdotV, sheenRoughness);
float sheenColor = sheenDistribution * sheenVisibility;
return sheenColor;
```
Thickness Texture

The thickness of a volume enclosed by the mesh is typically quite difficult to compute at run-time in a rasterizer. Since glTF is primarily used with real-time rasterizers, this extension allows for the thickness of the volume to be baked into a thickness map. Thickness is given in the coordinate space of the mesh. Any transformations applied to the mesh’s node will also be applied to the thickness. Thickness is an absolute value and defined as the product of thickness factor and thickness texture value. The thickness factor is defined in the range (0, inf), whereas the thickness texture value range is limited to [0, 1]. An exemplary configuration could set the thickness factor to the longest edge of the bounding box of a mesh, and the texture scales this value so that it corresponds to the actual thickness underneath each surface point.

Baking thickness into a map is similar to ambient occlusion baking, but rays are cast into the opposite direction of the surface normal. Dark values represent thin parts, bright values represent thick parts of the model.

Computing volumetric effects with a thickness map is a lossy process. We use the thickness map to estimate the distance that a light ray will travel until it exists the volume. The estimation takes place at the entrance point, where we fetch the thickness from the thickness map. The actual distance, however, varies with the angle of incidence at the entrance point, as this angle determines the direction into which the light ray will travel through the mesh.

Ray-tracers should ignore the thickness texture and use the actual, ray-traced distance instead. For this reason it is important that thickness factor and texture represent the actual thickness as accurate as possible. An accurate representation ensures that visual results look consistent across different rendering techniques. Note that it is still necessary to check the `thicknessFactor` to determine whether the object is thin-walled or volumetric.
```c
#define MATERIAL_TRANSMISSION

vec3 getIBLVolumeRefraction(vec3 n, vec3 v, float perceptualRoughness, vec3 baseColor, vec3 f0, vec3 f90, 
    vec3 position, mat4 modeMatrix, mat4 viewMatrix, mat4 projMatrix, float ior, float thickness, vec3 attenuationColor, float attenuationDistance) 
{
    vec3 transmissionRay = getVolumeTransmissionRay(n, v, thickness, ior, modeMatrix);
    vec3 refractedRayExit = position + transmissionRay;

    // Project refracted vector on the framebuffer, while mapping to normalized device coordinates.
    vec4 ndcPos = projMatrix * viewMatrix * vec4(refractedRayExit, 1.0);
    vec2 refractionCoords = ndcPos.xy / ndcPos.w;
    refractionCoords += 1.0;
    refractionCoords /= 2.0;

    // Sample framebuffer to get pixel the refracted ray hits.
    vec3 transmittedLight = getTransmissionSample(refractionCoords, perceptualRoughness, ior);

    vec3 attenuatedColor = applyVolumeAttenuation(transmittedLight, length(transmissionRay), attenuationColor, attenuationDistance);

    // Sample GGX LUT to get the specular component.
    float NdotV = clamp(dot(n, v), 0.0, 1.0);
    vec2 brdfSamplePoint = clamp(vec2(NdotV, perceptualRoughness), vec2(0.0, 0.0), vec2(1.0, 1.0));
    vec2 brdf = texture(u_GGXLUT, brdfSamplePoint).rg;
    vec3 specularColor = f0 * brdf.x + f90 * brdf.y;

    return (1.0 - specularColor) * attenuatedColor * baseColor;
}
#endif
```
Khronos glTF 2.0 Sample Viewer

This is the official Khronos glTF 2.0 Sample Viewer using WebGL: glTF 2.0 Sample Viewer

Homepage

github.com/KhronosGroup/glTF-Sample-Viewer
API

gltf Sample Viewer can be used without the web app, for example for integration into a thirdparty web application or for automated testing (see Render Fidelity Tools).

The API consists of several components that in combination allow flexible configuration of the gltf viewer.

More detailed information about the API is listed in the api documentation.

GltfView

The GltfView component is associated with one WebGL2 context. In practice this means it will be associated with one HTML5 Canvas. This component manages the interaction between the canvas and the GL context. For example it therefore specifies the viewport, the swapchain and can be used to schedule frame renders.

```javascript
const view = new GltfView(webGl2Context);
```

The view is also used to render frames, either on every window repaint event or on demand, e.g. when taking a frame capture.

```javascript
const update = () =>
{
    view.renderFrame(state, canvas.width, canvas.height);
    window.requestAnimationFrame(update);
}
Demo
Thanks!

Webinar
https://www.youtube.com/watch?v=JMiXv2PgGBY

Web App
https://github.khronos.org/gltF-Sample-Viewer-Release/

Source Code
https://github.com/KhronosGroup/gltF-Sample-Viewer

Contact us
@UX3DGpuSoftware
Ux3d.io