USD and glTF, a user's perspective

Eric Haines, NVIDIA

August 7, 2023

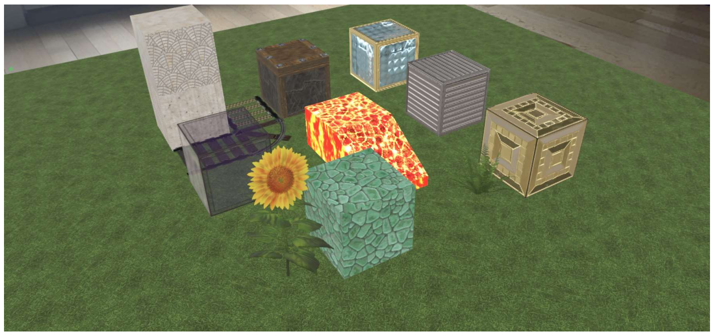
SIGGRAPH 2023

https://bit.ly/gltfusd

https://s2023.siggraph.org/presentation/?id=bof_153&sess=sess454

Play with your phone if you lose interest

On an iPhone (sorry, Android), view this model: https://bit.ly/mcusd10



What rendering bugs can you detect?

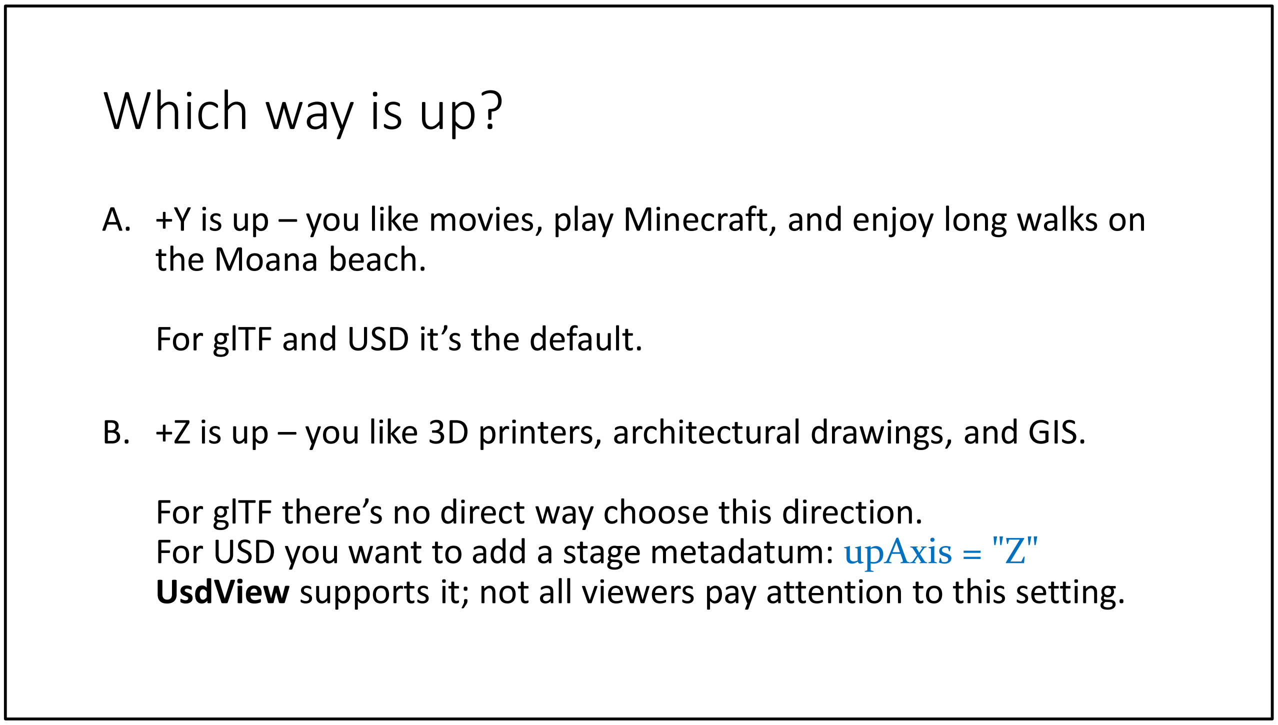
Apple Quick Look – other models here https://developer.apple.com/augmented-reality/quick-look/

Which way is up?

A. +Y is up in world space

B. +Z is up in world space

If you'd like to download these slides later: https://bit.ly/gltfusd or



Which way is up?

A. +Y is up – you like movies, play Minecraft, and enjoy long walks on the Moana beach.

   For glTF and USD it’s the default.

B. +Z is up – you like 3D printers, architectural drawings, and GIS.

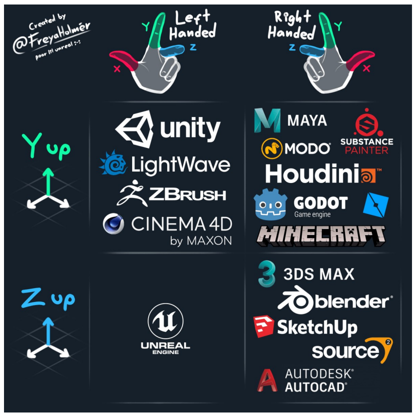
   For glTF there’s no direct way choose this direction.
   For USD you want to add a stage metadatum: `upAxis = "Z"`
   UsdView supports it; not all viewers pay attention to this setting.

A nice, easy way to get UsdView is https://www.nvidia.com/en-us/omniverse/ - free, no need to build it by hand etc.

Let’s not worry about left- and right-handed

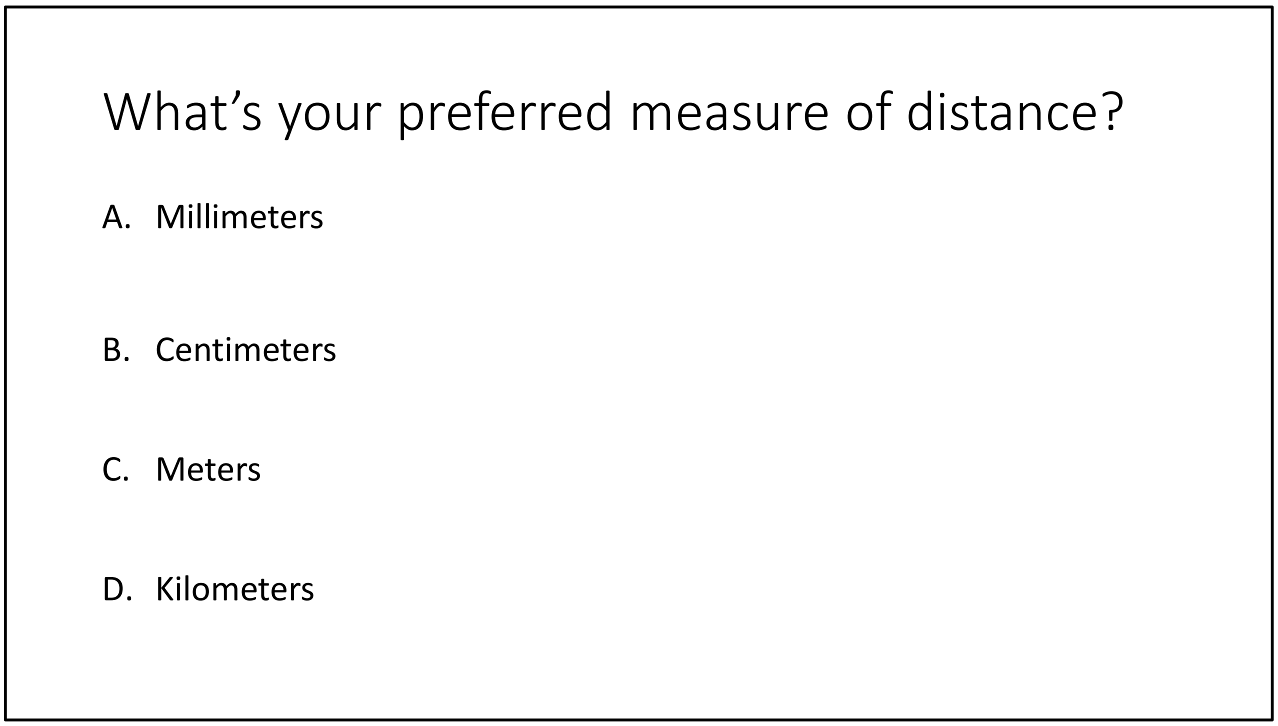
glTF and USD both use right-handed world coordinate systems.

Beware, though:



← I love that Minecraft is listed. Well, it’s arguably the most popular architectural modeling app in the world.

https://twitter.com/CasualEffects/status/1678263856802459648 is where I found this.



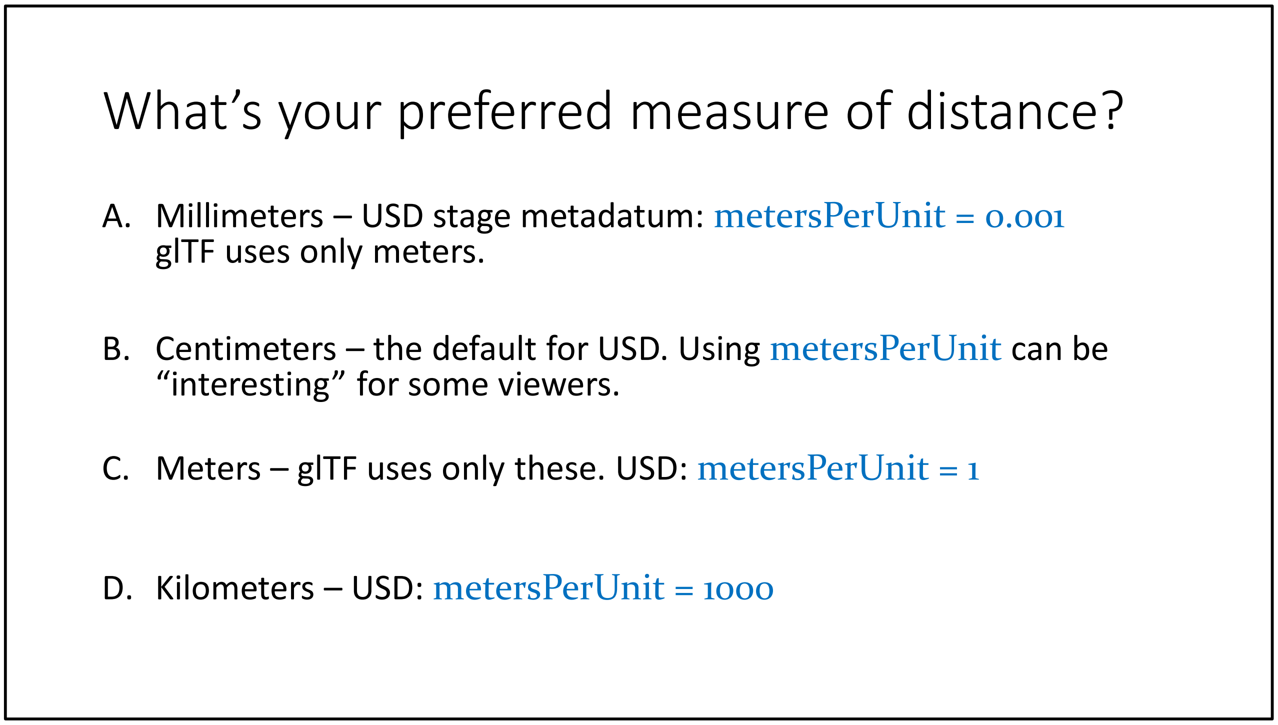
What’s your preferred measure of distance?

A. Millimeters

B. Centimeters

C. Meters

D. Kilometers



What’s your preferred measure of distance?

A. Millimeters – USD stage metadatum: metersPerUnit = 0.001
glTF uses only meters.

B. Centimeters – the default for USD. Using metersPerUnit can be “interesting” for some viewers.

C. Meters – glTF uses only these. USD: metersPerUnit = 1

D. Kilometers – USD: metersPerUnit = 1000

https://openusd.org/dev/api/group___usd_geom_linear_units__group.html
and https://docs.omniverse.nvidia.com/usd/latest/units.html

How do you set how much your camera sees?

A. Field of View angle (FOV)

B. Filmback and focal length

C. FOV and what the heck is filmback?

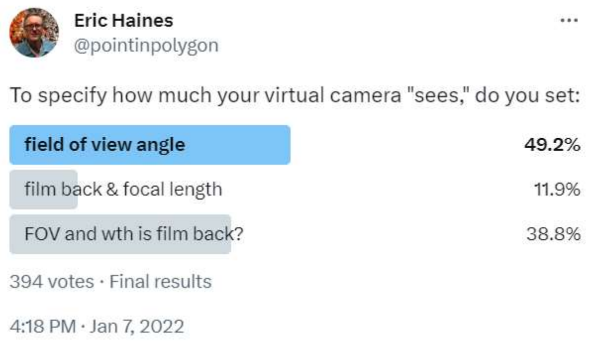
How do you set how much your camera sees?

A. Field of View angle (FOV)

B. Filmback and focal length

C. FOV and what the heck is filmback?

Don’t care; it’s easy to convert



Eric Haines
@pointinpolygon

To specify how much your virtual camera "sees," do you set:

field of view angle	49.2%
film back & focal length	11.9%
FOV and wth is film back?	38.8%

394 votes · Final results

4:18 PM · Jan 7, 2022

glTF uses vertical FOV (yfov)

USD uses filmback and focal length

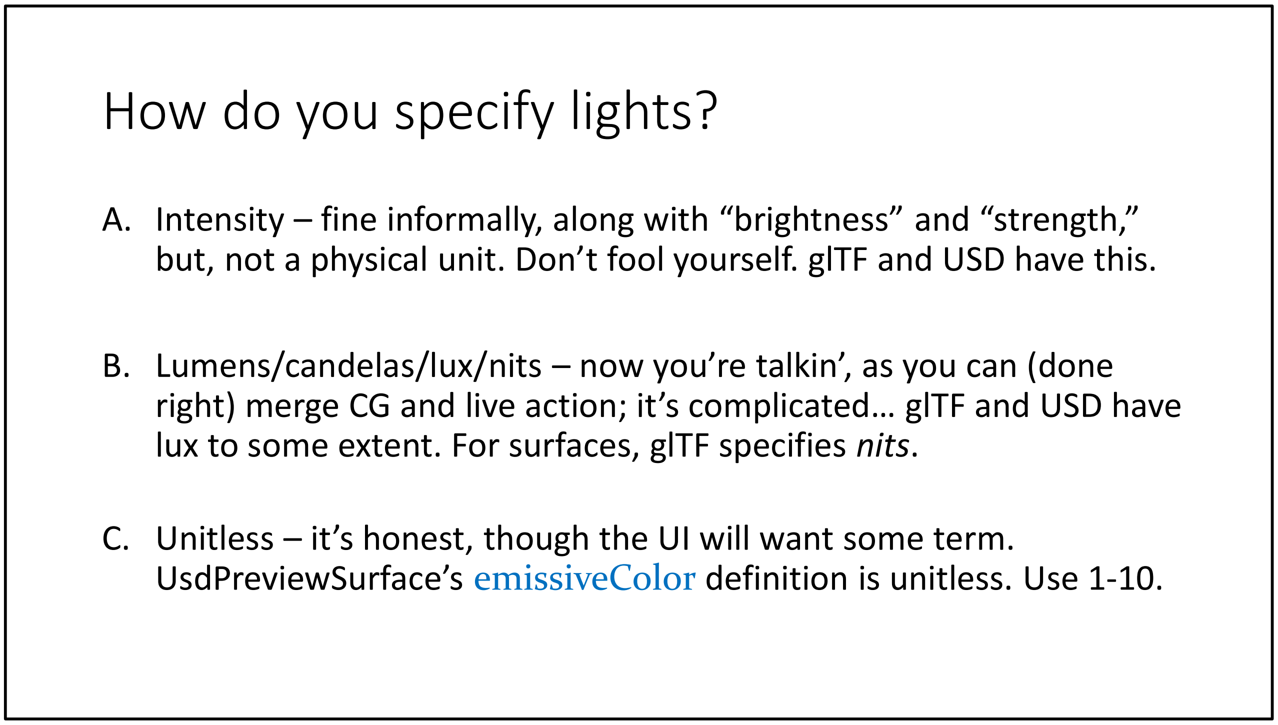
https://twitter.com/pointinpolygon/status/1479563199955488776 for poll - USD uses filmback and glTF uses vertical FOV, yfov

How do you specify lights?

A. Intensity

B. Lumens/candelas/lux/nits

C. Unitless



How do you specify lights?

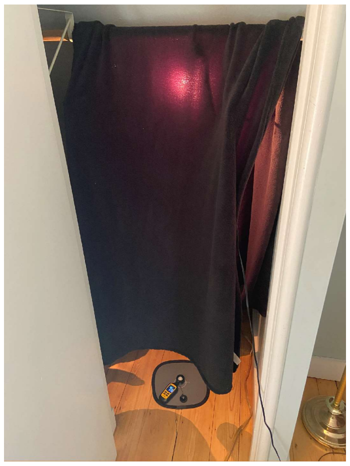
A. Intensity – fine informally, along with “brightness” and “strength,” but, not a physical unit. Don’t fool yourself. glTF and USD have this.

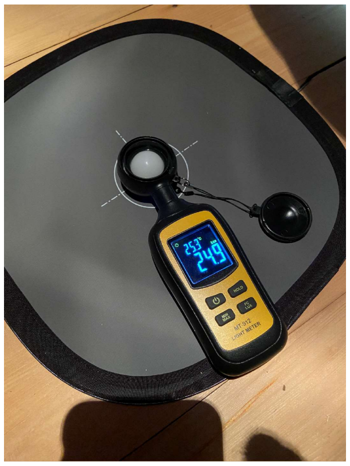
B. Lumens/candelas/lux/nits – now you’re talkin’, as you can (done right) merge CG and live action; it’s complicated… glTF and USD have lux to some extent. For surfaces, glTF specifies *nits*.

C. Unitless – it’s honest, though the UI will want some term. UsdPreviewSurface’s emissiveColor definition is unitless. Use 1-10.

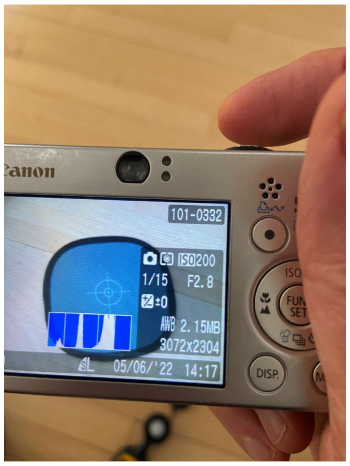
https://openusd.org/release/spec_usdpreviewsurface.html

Maybe don’t be me


The closet “studio”

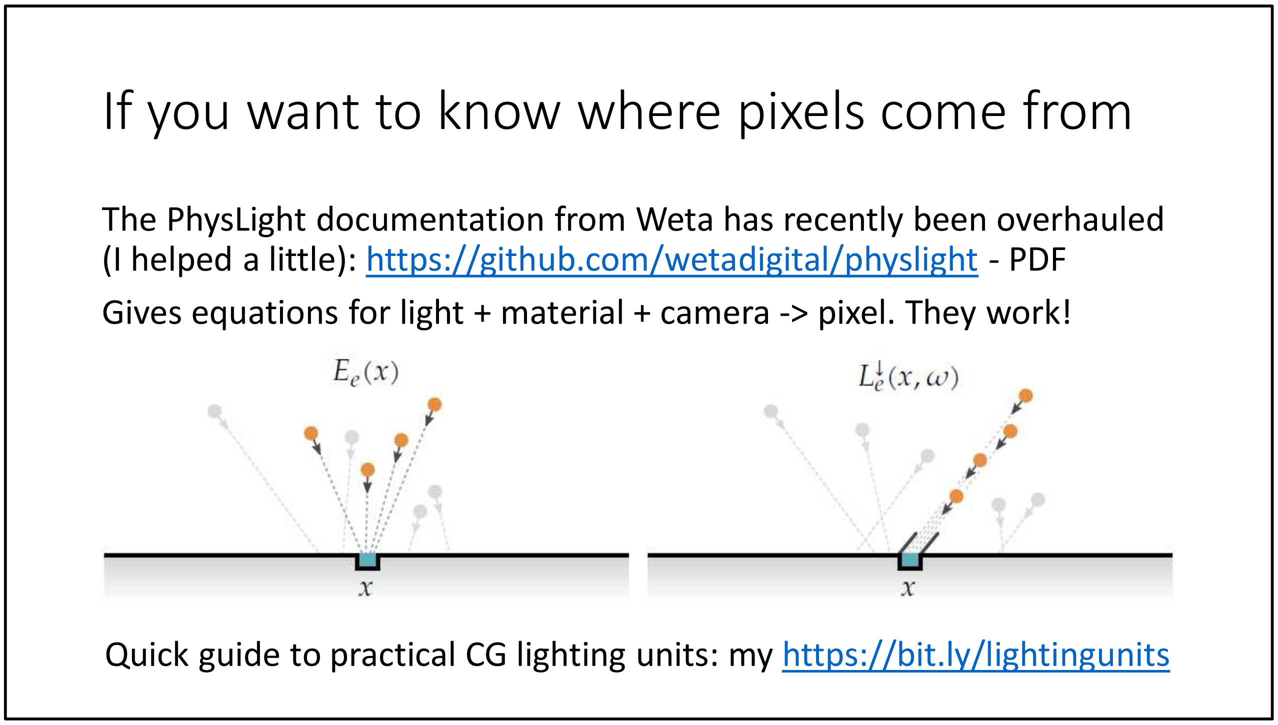


The fine $29.99 illuminance meter



An old camera shows some data

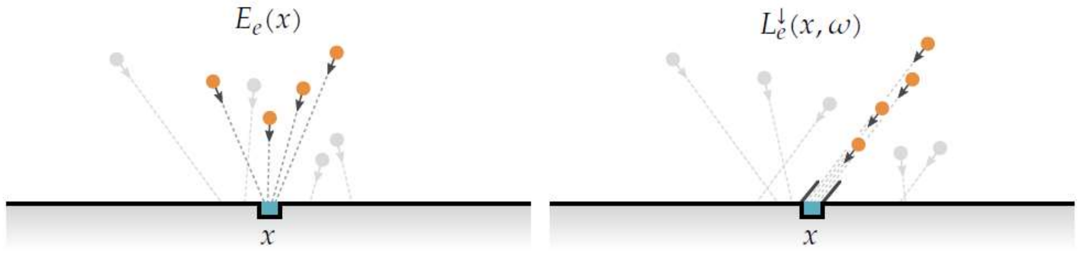
Cheapo meter at https://www.amazon.com/gp/product/B075DC6X25 - I find the recessed dome a little suspect



If you want to know where pixels come from

The PhysLight documentation from Weta has recently been overhauled (I helped a little): https://github.com/wetadigital/physlight - PDF

Gives equations for light + material + camera -> pixel. They work!



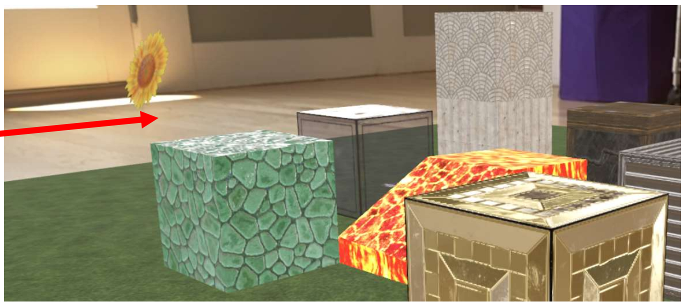
Quick guide to practical CG lighting units: my https://bit.ly/lightingunits

Well, they mostly work. The iPhone in daylight seemed a fair bit off. At https://github.com/wetadigital/physlight and easy guide at https://bit.ly/lightingunits

The ~~bugs~~ unimplemented features: sidedness

For this USD scene, single vs. double sided is not respected.

No stem from this angle

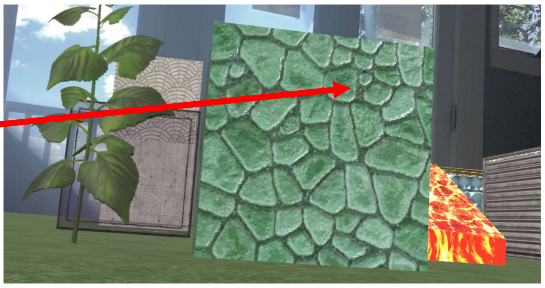


- In glTF, you put in the *material*: "doubleSided": true,
- In USD, you put in the *Mesh*: bool doubleSided = 1

This is one where the viewer should really work properly. The workaround of "let's create a separate backside polygon", so each thing has two polygons back-to-back, can cause z-fighting artifacts – my users complained.

The ~~bugs~~ unimplemented features: normals

For this USD scene, normal maps are not adjusted properly:

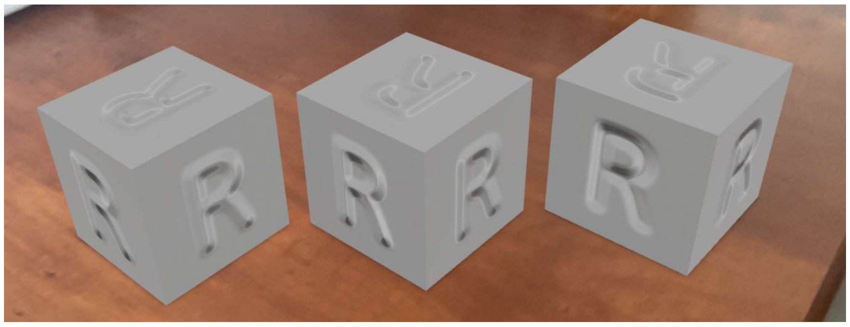


- In glTF, you have to fix the normal map itself. OpenGL-style only.
- In USD, you can negate the Y component’s bias and scale. DirectX OK.

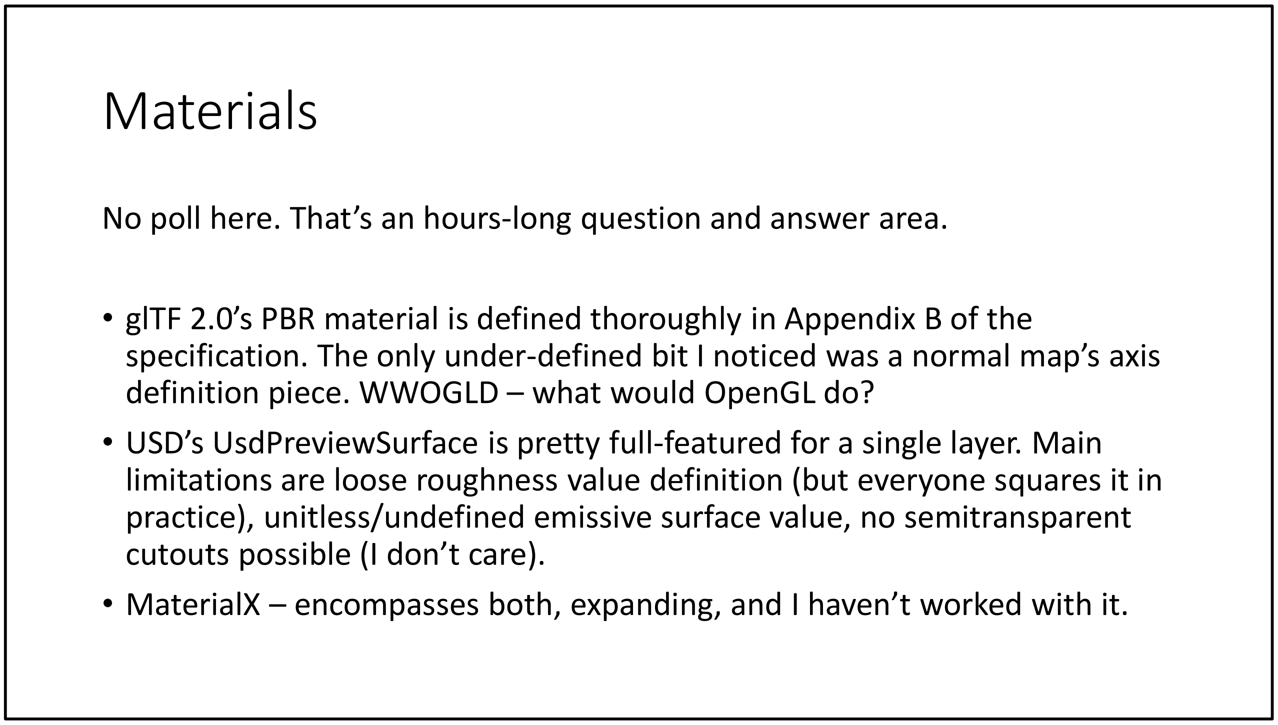
Making the subtle unsubtle

See the ASWF’s USD Working Group assets github repo:
https://github.com/usd-wg/assets

Two types of models there: “full” and “test”. See the “test” directory NormalsTextureBiasAndScale – these *should* all look the same:



https://github.com/usd-wg/assets and https://github.com/usd-wg/assets/tree/main/test_assets/NormalsTextureBiasAndScale

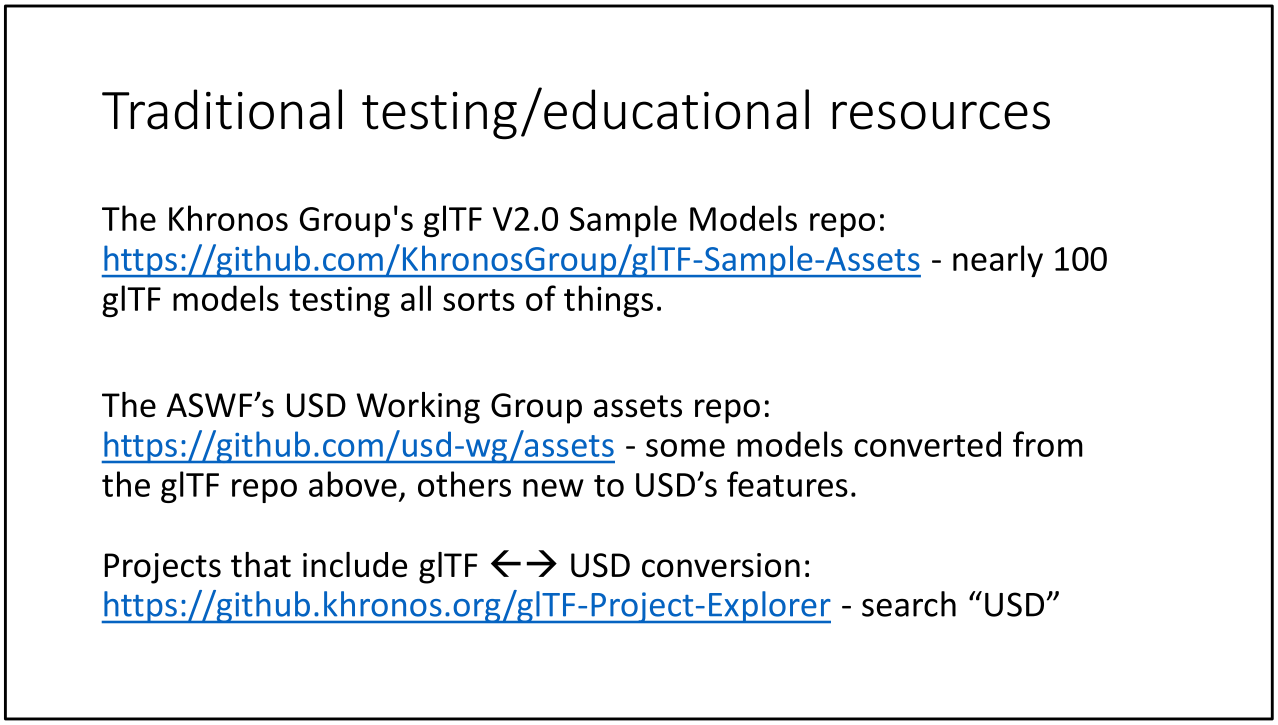


Materials

No poll here. That’s an hours-long question and answer area.

- glTF 2.0’s PBR material is defined thoroughly in Appendix B of the specification. The only under-defined bit I noticed was a normal map’s axis definition piece. WWOGLD – what would OpenGL do?
- USD’s UsdPreviewSurface is pretty full-featured for a single layer. Main limitations are loose roughness value definition (but everyone squares it in practice), unitless/undefined emissive surface value, no semitransparent cutouts possible (I don’t care).
- MaterialX – encompasses both, expanding, and I haven’t worked with it.

https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html
and https://openusd.org/release/spec_usdpreviewsurface.html
and https://materialx.org/DeveloperReference.html



Traditional testing/educational resources

The Khronos Group's glTF V2.0 Sample Models repo:
https://github.com/KhronosGroup/glTF-Sample-Assets - nearly 100 glTF models testing all sorts of things.

The ASWF’s USD Working Group assets repo:
https://github.com/usd-wg/assets - some models converted from the glTF repo above, others new to USD’s features.

Projects that include glTF ←→ USD conversion:
https://github.khronos.org/glTF-Project-Explorer - search “USD”

ASWF group also has meetings and a Slack server (I’m guessing glTF does, too, but I have no time for that)

Other resources and SIGGRAPH stuff

Tools such as usdchecker are your friends. https://openusd.org/release/toolset.html

NVIDIA Omniverse is based on USD and has a few free tools available:

- USD Composer (aka Create) reads and writes glTF and USD files.
- It also includes an asset validator, https://docs.omniverse.nvidia.com/extensions/latest/ext_asset-validator.html

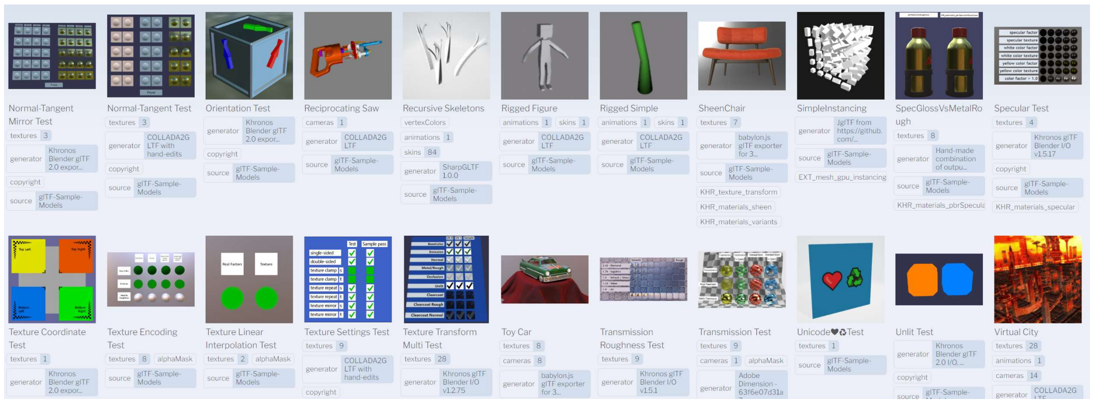
See https://wiki.aswf.io/display/WGUSD/Siggraph+2023 for a list of all USD-related sessions and events at SIGGRAPH.

See https://www.khronos.org/events/2023-siggraph for glTF at SIGGRAPH.

NVIDIA keynote 8 AM tomorrow morning in Hall K will include some USD validation announcements.

New resource: “Explore 3D Assets”

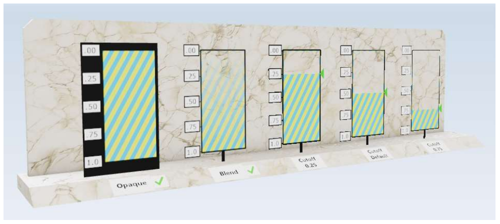
https://asset-explorer.needle.tools by Felix Herbst. 64 models to start.



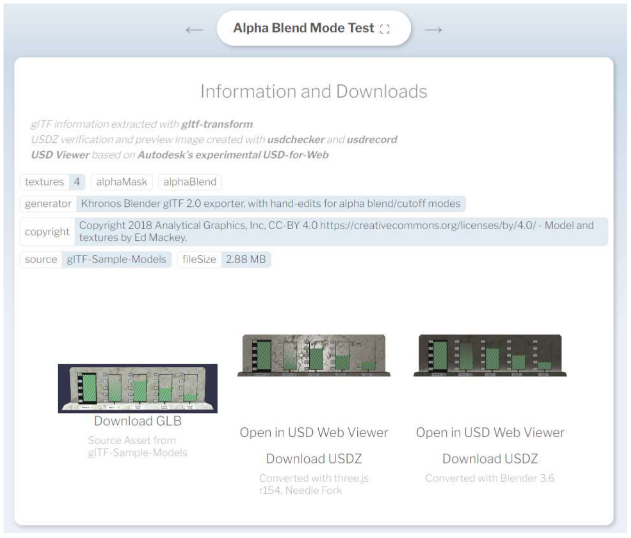
Many models from https://github.com/KhronosGroup/glTF-Sample-Assets, as a start

Felix Herbst’s announcement:
Hi all! I'm happy to announce **Asset Explorer**, a side-project-turned-useful that I think can benefit both the USD and glTF communities.
•Lists and displays assets and which features they use
•currently the ones from glTF-Sample-Models
•Has automatic conversions to USDZ, one with three.js and one with Blender 3.6
•I plan to run conversions again on meaningful updates to either.
•There's *lots* of known issues here, but I wanted to show the current state of things. This will get better!
•Generated USDs are automatically rendered with *usdrecord* and checked with *usdchecker*
•Allows directly opening the USD files in the web
•uses Autodesk's USD-WASM + three.js Hydra delegate + fixes by me
•Supports viewing on the web, iOS QuickLook and visionOS to test out these files superfast.
I hope this effort can help to accelerate the great work that's already going on to improve the USD spec, improve compatibility between USD applications, improve compatibility with the glTF world.You find Asset Explorer here: https://asset-explorer.needle.tools/
Please let me know if there's any questions!

The Alpha Blend Mode Test



Interactive model in the browser, renderers based on three.js



Alpha Blend Mode Test

Information and Downloads

*glTF information extracted with **gltf-transform**.*
*USDZ verification and preview image created with **usdchecker** and **usdrecord**.*
USD Viewer** based on **Autodesk's experimental USD-for-Web

textures	4
alphaMask	
alphaBlend	
generator	Khronos Blender glTF 2.0 exporter, with hand-edits for alpha blend/cutoff modes
copyright	Copyright 2018 Analytical Graphics, Inc, CC-BY 4.0 https://creativecommons.org/licenses/by/4.0/ - Model and textures by Ed Mackey.
source	glTF-Sample-Models
fileSize	2.88 MB

Download GLB
Source Asset from glTF-Sample-Models

Open in USD Web Viewer
Download USDZ
Converted with three.js r154, Needle Fork

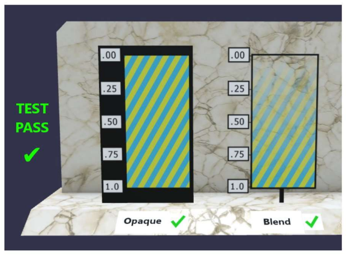
Open in USD Web Viewer
Download USDZ
Converted with Blender 3.6

This model is at https://github.com/KhronosGroup/glTF-Sample-Assets/tree/main/Models/AlphaBlendModeTest

The Alpha Blend Mode Test, continued

Based on the many test models and related debug information at https://github.com/KhronosGroup/glTF-Sample-Assets

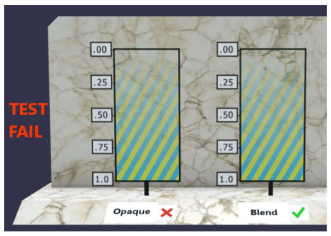
`OPAQUE` vs `BLEND`



The box on the far left uses the default alphaMode, `OPAQUE`. Although the texturemap supplies an alpha channel, the alpha values are intentionally ignored here.

The next box shows the effects of alpha blending. The texture contains a linear alpha ramp inside a black border, along with some labels. At the bottom there are green check marks and hidden red X's, that will show check marks only when the requested mode is correctly applied.

Problem: Alpha Values Used in Opaque Mode



The above screenshot shows what typically happens if a rendering engine decides to process alpha values from a texturemap without blending being specifically requested. This is a test failure, and should be corrected in the engine.

Note that a red "X" mark has appeared next to `OPAQUE`, due to a green checkmark with zero alpha values being blended away.

This model is at https://github.com/KhronosGroup/glTF-Sample-Assets/tree/main/Models/AlphaBlendModeTest

Test, Test, Test (and, Contribute!)

I've put this presentation up at:

https://bit.ly/gltfusd

Read the notes for the slides, too.

Me, I'm erich@acm.org

http://erichaines.com

@pointinpolygon on twit-excuse-me-X