Hi my name is Eric Chadwick and I’m going to talk about how creatives can take advantage of KTX2 for texture compression.
A bit about me. I’m a Staff Technical Artist here at Wayfair, working as a technical director for both our realtime 3D and offline rendering pipelines. With Khronos I’m a regular contributor in the 3D Commerce and 3D Formats working groups, collaborating on standards, making sample models, and developing tutorials and documentation. These are a few of the sample models I’ve worked on, which you can find here on GitHub.

https://github.com/KhronosGroup/gltf-Sample-Models
KTX2 provides an efficient way to compress and transmit your textured assets. You get huge savings and you can preserve the original quality of your assets. Let’s look at a live comparison.
Special thanks to Gary Hsu at Microsoft for putting this demo together in Babylon.js, and to Richard Sahlin at IKEA for fine-tuning the compression settings. On the left is the original model, with JPG and PNG textures. And on the right is the version with KTX2 textures. At normal viewing distance, they’re virtually indistinguishable. At Wayfair we’re coming at this from an e-commerce perspective; how do our customers normally see these models? At this distance, there’s really no difference. And even if we zoom in close, the differences are really hard to spot. We’re getting really high-fidelity results, for about a fifth of the GPU memory cost. And the transmission size is also a bit smaller. Wow, I’m impressed.

live demonstration

(https://playground.babylonjs.com/full.html#YD2TXP#23)
So how does it work? You provide a glTF model with JPG and/or PNG textures. That’s the cylinders on the left. You encode them into KTX2, using a form of compression that’s been generalized to work across platforms. When a viewer loads the model, the textures are transcoded live to whichever GPU format is needed for their particular device. KTX2 textures stay compressed in GPU memory. Pretty cool. You get better results if you can feed it lossless PNG textures, although you can also feed it JPGs. Another handy tip is if you want high quality compression results, well, it’s going to take longer to compress those. But it’s worth it.
Yeah so why is this great? Here are some simple graphs made in Google Sheets; I’ll show you how you can get this kind of data a bit later. The key thing to notice is how inefficient JPG and PNG formats are, once you load the asset onto the GPU to render it. Notice the tall red bar in the chart on the left. Those textures have to be uncompressed completely into 32bit files; they’re non-native to the GPU. However KTX2 is transcoded directly into GPU-supported codecs, and stays compressed. Memory is a spare resource on many devices. Especially if you want to load multiple assets at once, say for a 3D room planner, or for a game. On the right is a graph showing a common ask, why can’t you just downsize all the textures instead of
using KTX2? If you halve the resolution you’re not really getting the same results as you get with KTX2. Sure the file size is smaller, but the GPU size is larger, and don’t forget all your textures are a lot blurrier too, because you’re using half-resolution JPGs and PNGs. The KTX2 files are still at their original resolutions here. So usually it’s not worth it to just halve them and not supercompress.
KTX2 gives you two main compression codecs. ETC1S is the smaller, and it excels at textures with homogenous colors. What does that mean? Think photographic content, or typical color maps. It’s similar to the way JPG works better on photos than it does on graphics. When the RGB color channels have similar values, ETC1S does a great job.

UASTC is a bit larger, but you need it for textures with lots of variation across the color channels. ORM for example; which packs Occlusion, Roughness, and Metalness into three RGB channels; with ORM the channels are usually quite different. This is a challenge for ETC1S, as we’ll see in a bit. But UASTC does a great job, at the cost of a slightly larger size.
Let’s Make KTX2!

So let’s compress some textures. Fortunately we have a few different tools at our disposal. On the left we have some great graphical tools that really help demystify the process, and give you instant visual feedback. On the right we have command-line tools, which give you the ultimate control. It’s really up to you which work best in your situation. Also there’s a call to action here, we want to hear what tools you want to see. So, please get in touch with the working groups on Slack and let us know!

Future tools?
Khronos wants to hear from you!
What tools do you want to see?
khronosdevs.slack.com
Let’s run through the process with glTF-Transform. This is a command-line tool, without a pretty GUI, but it’s really pretty easy to use. A shout-out to Don McCurdy here, for making this great tool! It does a bunch of things for glTF assets, but today we’re just interested in KTX2. So here’s a quick command for converting a GLB’s textures into KTX2. The command line may be a bit daunting for creatives if you haven’t used it before. But it’s really pretty simple, and as we’ll see it’s pretty powerful too.
So let’s open up a Command prompt. I’m on Windows, so it’s as simple as holding the Windows key and pressing R (for RUN). Then type CMD and hit Enter. Let’s full-screen this window, and hold Ctrl and use the mouse wheel to zoom in a bit. First let’s switch to the folder where my files are. CD for change directory. Then the path where my sample files are. DIR shows me what’s in this folder. The input.glb file is the one we want to compress, and the others are some sample compressions. Let’s enter the commands for compressing to KTX2. gltf-transform is the tool. etc1s is the codec. Then the file we’re going to compress, and the file we’re going to save out to. We’re going to add a --quality 255 setting, which tells it to use the highest quality
(at the expense of speed). And we’ll use **--verbose** to tell it to be loud about it, we want to see all the steps it performs. Then we press Enter to run it. It only takes about 25 seconds, not too bad.
Let’s take a look at the results, and compare with the source file. There are a bunch of glTF renderers out there, I’m just going to show the Khronos viewer, it’s all we really need at this point, and it has some nice features.
Let’s load the original source model. Then duplicate the window, and load the compressed file to compare. At this distance we can use the DEBUG views to examine individual textures, and compare differences. Base color, normal, occlusion, roughness, metalness.

live demonstration

(https://github.khronos.org/glTF-Sample-Viewer-Release/)
So having seen what ETC1S does with ORM and Normal maps, let’s twist some dials to get higher quality results. To do this, we’ll do the compression in stages, split by texture type. You can see there’s three lines of commands here. (click) For the first step we’ll use UASTC, and we’ll limit it to just running on the Normal Maps. The level 4 setting produces the highest quality, it just takes longer to compress. The rdo .5 setting controls how much LZ compression to apply; here it’s a medium-strength setting, a good tradeoff between quality and size. The slots setting confines this step to only apply to the Normal Maps inside the glTF file. There’s a command called inspect you can use to see all the slot names. You can learn about all these settings in
the KTX Artist Guide. At the end of the line, the `zstd 18` setting is how much supercompression to use. (click) In the second step, we start with the GLB that’s output by the first step, and apply a different RDO value, and only to the ORM maps. (click) In the third step, we simply use ETC1S for all the remaining textures. glTF-Transform is smart about this, if a GLB has a some KTX2 textures already it will skip those, and only compress the other textures. This takes about 7 minutes to run, so I’m not going to demo it here. But you get the picture, we’re sacrificing compression speed to get higher quality results.
Let’s Compare (UASTC+ETC1S)

- ETC1S = smaller file, more noise
  good with homogenous channels
- UASTC = a bit larger, less noise
  good for disparate channels

Just looking at the File Explorer, we can see right away the file size is larger with the high-quality settings than with ETC1S alone. It’s about 8mb versus about 4.5mb. But the quality is quite a bit better, as we’ll see. Let’s load the results in the Sample Viewer to compare channels again.
Let’s load the original source model. Then duplicate the window, and load the compressed file. If we look at the Debug channel for Roughness, we can see there are much fewer artifacts than we had with ETC1S alone. The nice thing is, if you spend a bit more time fine-tuning the compression settings, you can get virtually indistinguishable results, as we showed earlier in the side-by-side demo. You can find the full settings for that compression in the KTX Artist Guide.

live demonstration
(https://github.khronos.org/gltf-Sample-Viewer-Release/)
A nice thing with glTF-Transform is the `inspect` command (click), which prints out a bunch of useful info about your glTF file. If you add `format csv` you can get comma-separated values, which are easily loaded into a spreadsheet. The more-than symbol at the end there will save the data out to a file, which you can name however you like... `info.txt` or `output.csv` it doesn’t matter it’s just a text file. If you load that file into a spreadsheet like Sheets or Excel, it will format it into columns and rows, and you can use formulas to get totals. You can even graph it out, here’s a simple Google Sheet graph comparing the `inspect` data for three glTF files. Graphs are really useful to help people understand the relationships between all these values.
That’s basically it. Here are links to all the resources I’ve mentioned. In the Khronos 3D Formats working group we wrote the KTX guides, to walk through the whole process and explain all the commands. I also want to say, at Wayfair we’re pretty excited about what KTX2 brings to the whole process; we’re really looking forward to using it. It’s an exciting time to be using glTF! Thanks for your time.