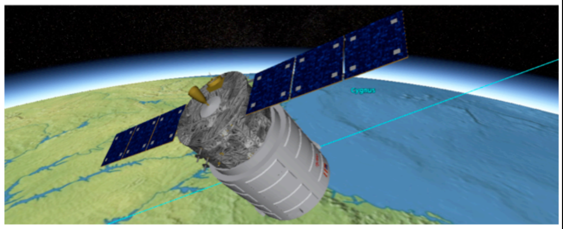
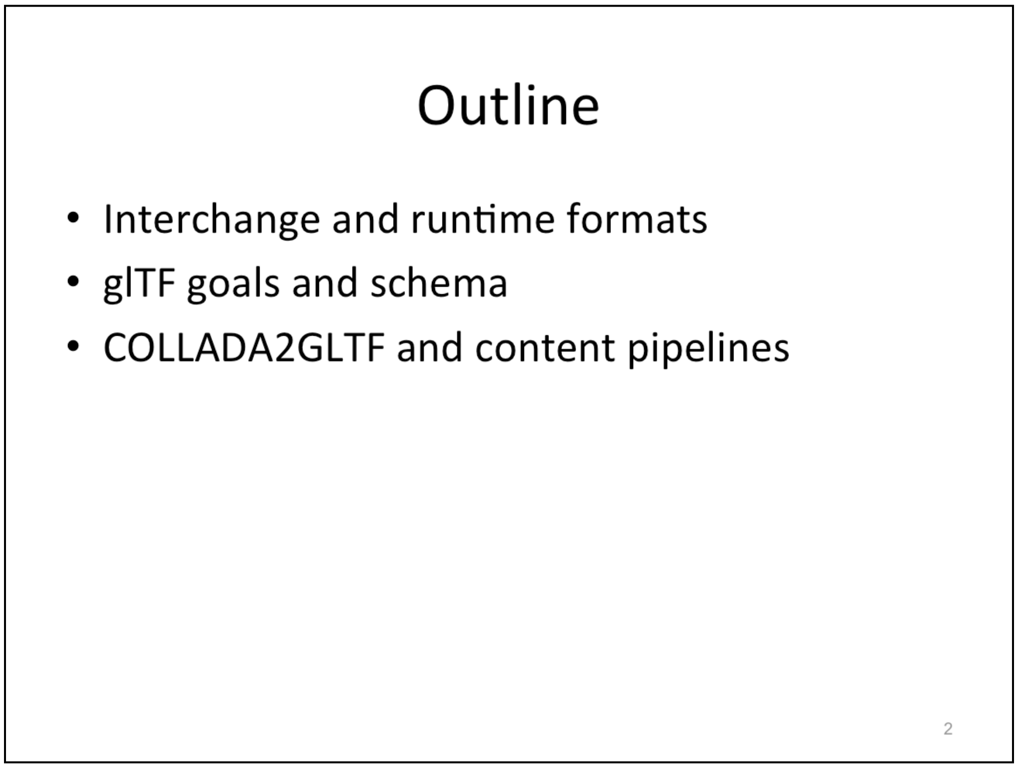
WebGL Content Pipeline with glTF



Patrick Cozzi, @pjcozzi
Analytical Graphics, Inc.
University of Pennsylvania

Like OpenGL, WebGL is a rendering API that exposes the capabilities of the hardware. It knows about low-level concepts like buffers, textures, shader programs, and uniforms. Artists, on the other hand, use modeling tools like Maya or Modo, to create assets using much higher-level constructs such as geometries, node hierarchies, materials, and animations. As engine developers, it is up to us to create a content pipeline that brings assets from modeling tools to our WebGL-based engines. Furthermore, this pipeline needs to produce runtime assets that are easy and efficient to use on the web with WebGL.

Historically engine developers have created custom asset formats for their engine and custom exporters for modeling tools or converters from interchange formats like COLLADA. This talk introduces glTF, the runtime asset format for WebGL, OpenGL ES, and OpenGL, which significantly reduces the amount of work engine developers have to do by providing an efficient and extensible format based on JSON and binary blobs, and an open-source content pipeline for creating glTF assets from COLLADA.

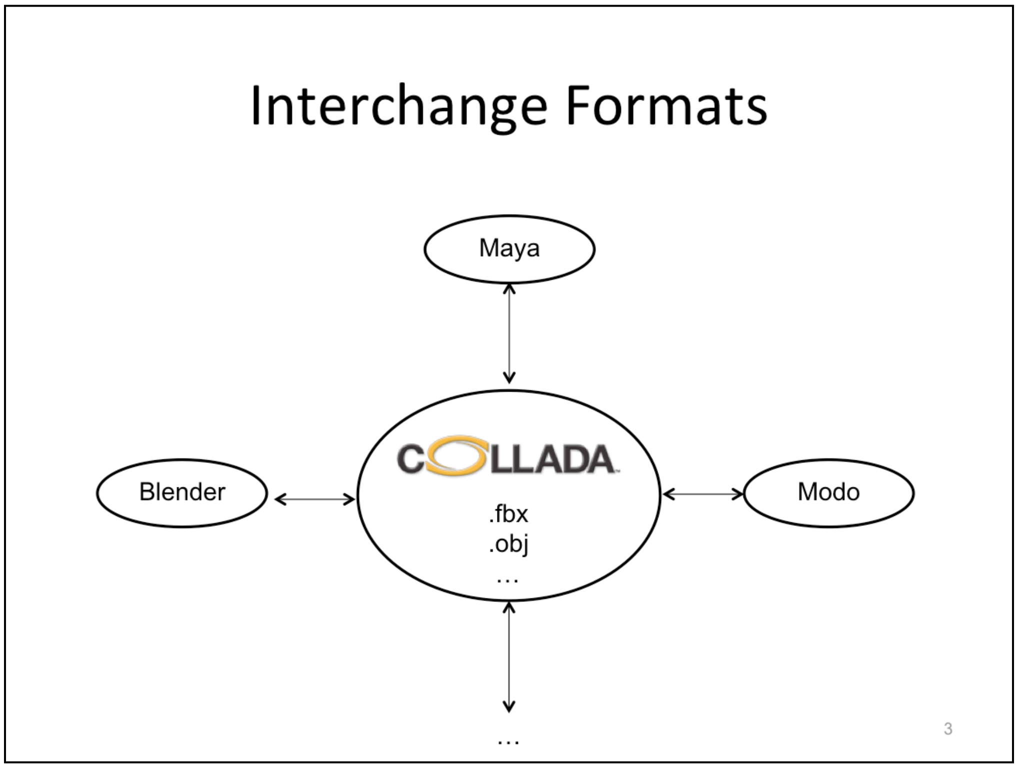


Outline

- Interchange and runtime formats
- glTF goals and schema
- COLLADA2GLTF and content pipelines

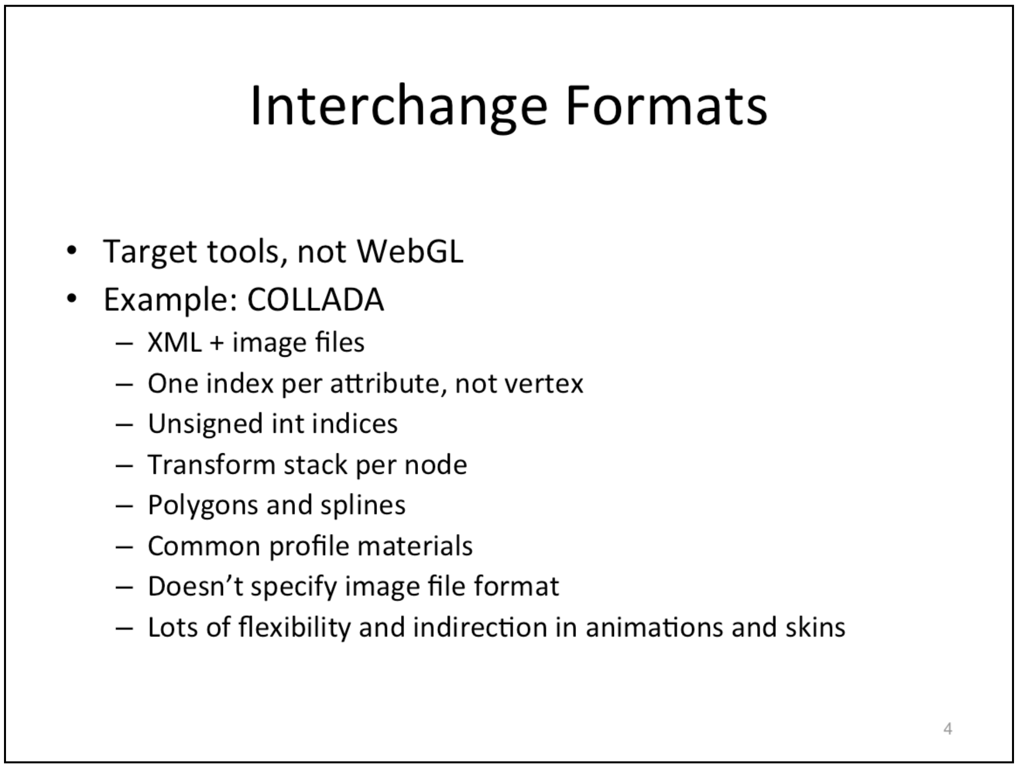
2

We'll briefly talk about the differences between interchange and runtime formats. Then, we'll look at the design goals of glTF and some examples of the schema. Finally, we'll look at the open-source COLLADA-to-glTF converter and how it fits into a content pipeline.



Interchange Formats

Interchange formats are used by artists to move assets from one tool to another. These formats strive to preserve the full authoring contents of an asset but are not optimized for runtime engines, especially for use on the web with WebGL.



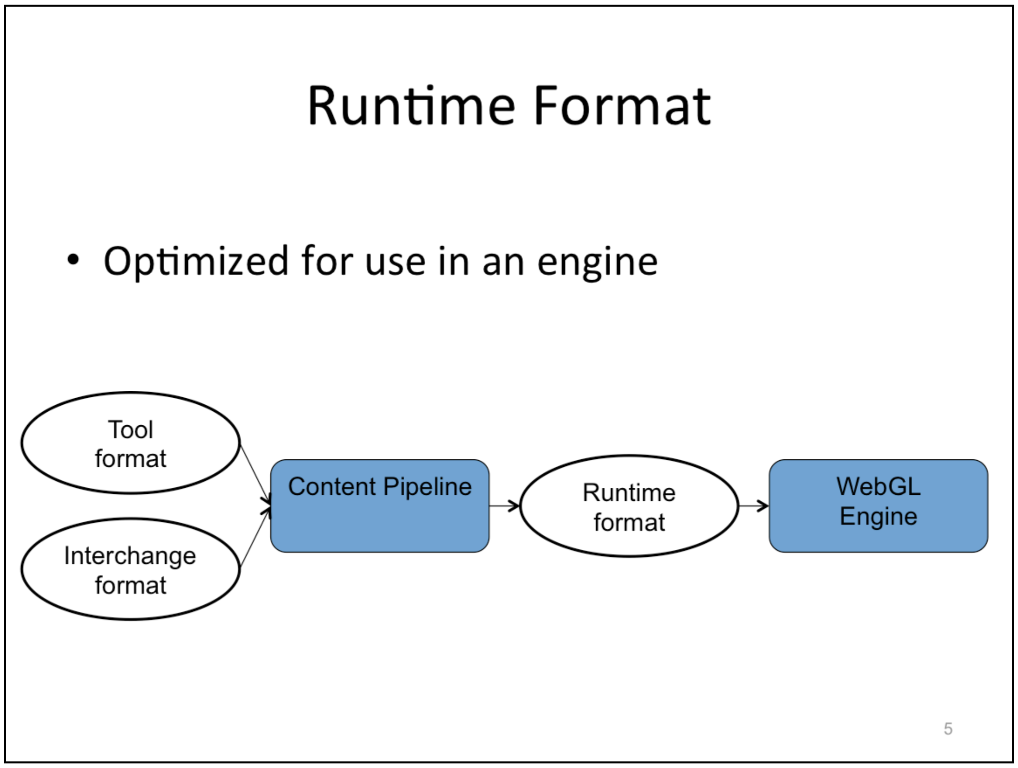
Interchange Formats

- Target tools, not WebGL
- Example: COLLADA
  - XML + image files
  - One index per attribute, not vertex
  - Unsigned int indices
  - Transform stack per node
  - Polygons and splines
  - Common profile materials
  - Doesn't specify image file format
  - Lots of flexibility and indirection in animations and skins

4

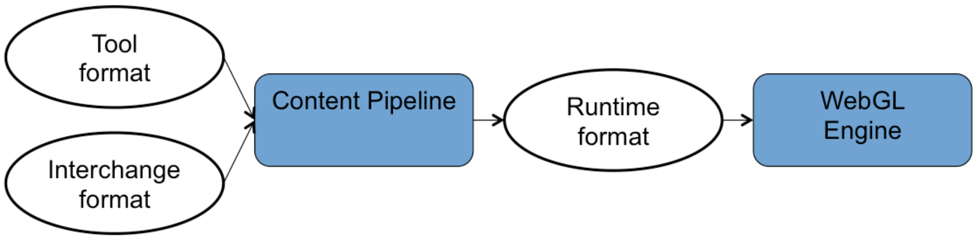
Interchange formats are generally verbose and slow to load for runtime use. They need to go through many conversion steps before being ready for WebGL. This processing doesn't belong in the runtime engine; it belongs in the content pipeline that runs offline or in the cloud.

Several examples from COLLADA are shown here. XML is verbose; indices are specified per attribute, not per vertex as WebGL requires; unsigned int indices are used, which requires a WebGL extension (a widely supported one though) or division into multiple meshes; each node can have a stack of transforms, whereas at runtime, we just want a 4x4 matrix; polygons and splines need to be tessellated; shaders need to be generated for materials (however, some engines will want to do this at runtime to match their g-buffer format for deferred shading, for example); any image format may be used, including ones not supported by browsers; and keyframe animations support several different splines, which is more flexibility than most runtime engines need.



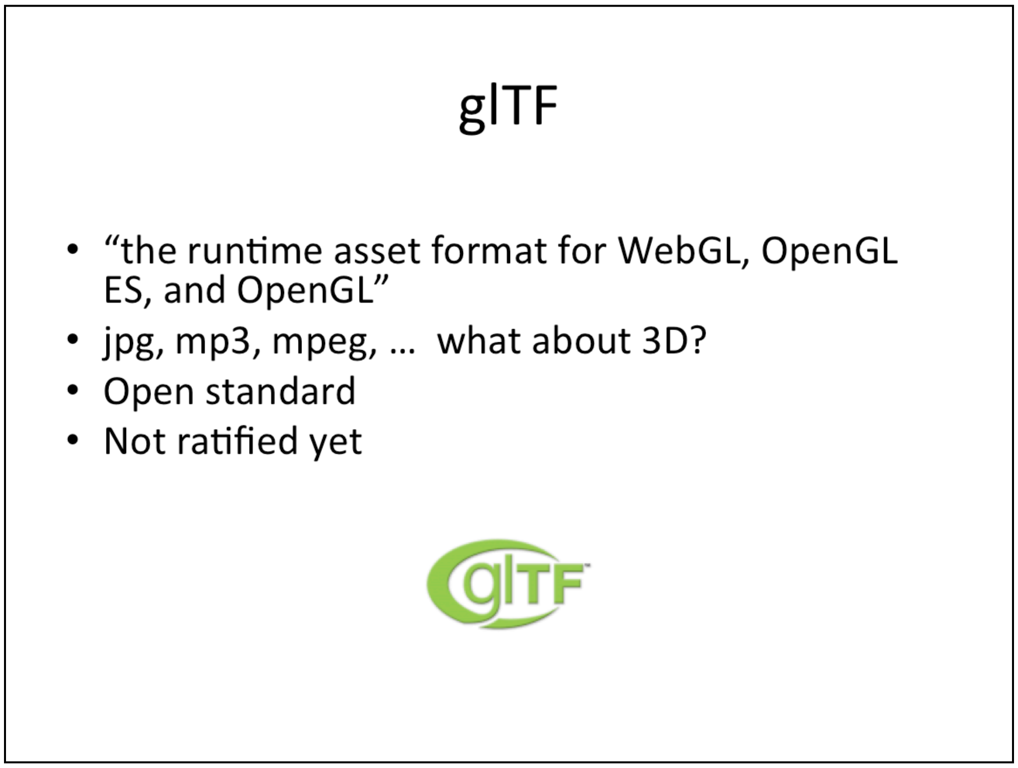
Runtime Format

- Optimized for use in an engine



5

Instead of loading interchange formats, most engines use a content-pipeline to convert assets to a runtime format for their engine. Thinking in terms of images, this is similar to creating an image as a .bmp but then publishing it to the web as a .jpg.



glTF

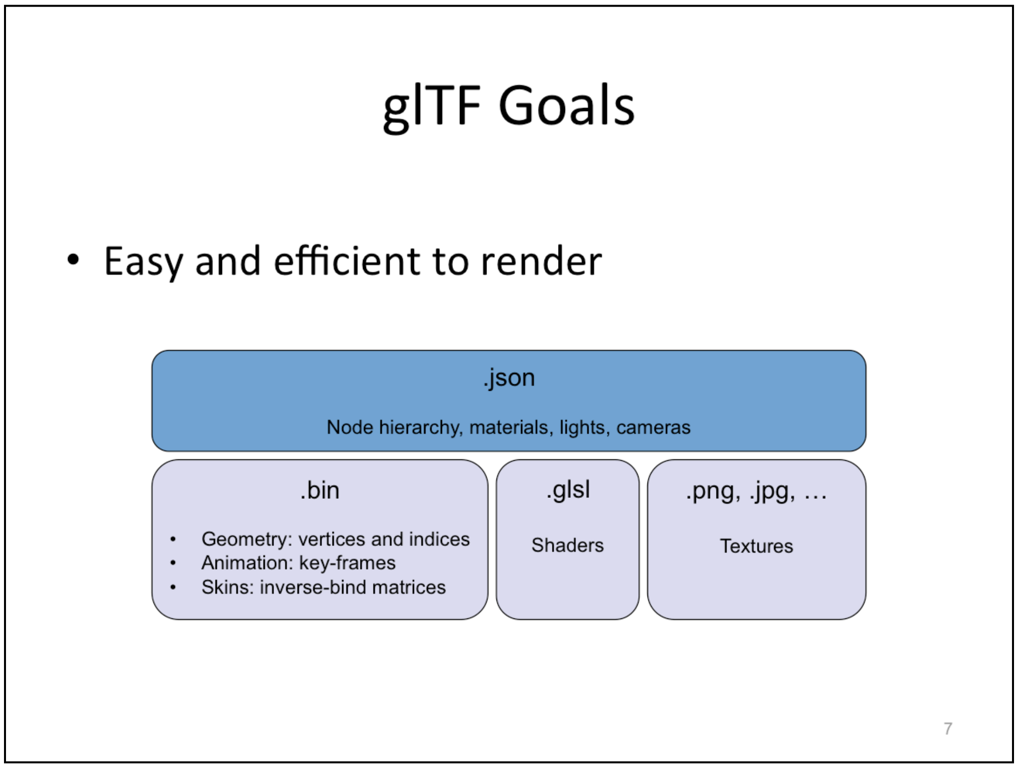
- “the runtime asset format for WebGL, OpenGL ES, and OpenGL”
- jpg, mp3, mpeg, … what about 3D?
- Open standard
- Not ratified yet

With WebGL, there are a handful of ad-hoc runtime formats, usually based on JSON, for a few of the major engines like Three.js [1] and Babylon.js [2].

glTF is an open-standard runtime asset format that strives to be useful to most WebGL engines by providing a schema and content pipeline for the most common asset constructs like geometry and materials, and to be extensible so each engine can add their own custom data.

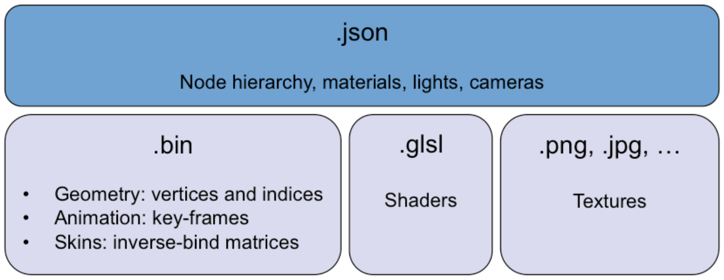
[1] https://github.com/mrdoob/three.js/wiki/JSON-Model-format-3.1

[2] https://github.com/BabylonJS/Babylon.js/wiki/Babylon.js-file-format



glTF Goals

- Easy and efficient to render


.json
Node hierarchy, materials, lights, cameras
.bin
Geometry: vertices and indices
Animation: key-frames
Skins: inverse-bind matrices
.glsl
Shaders
.png, .jpg, …
Textures

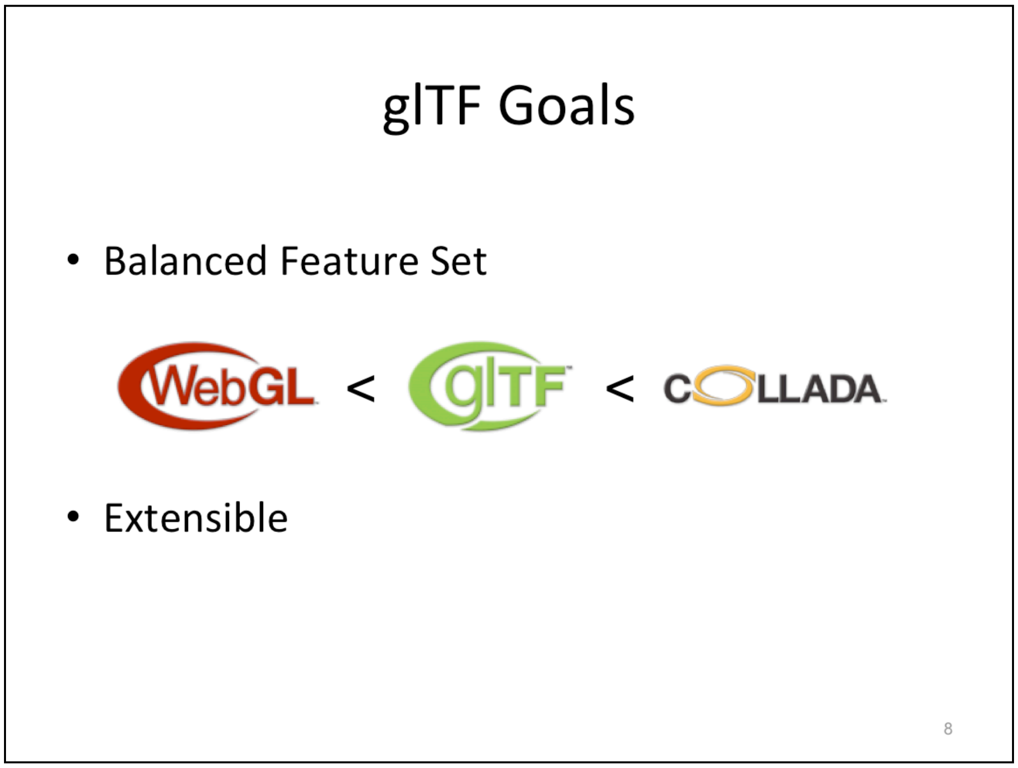
7

The number one goal of glTF is that assets are easy and efficient to render in WebGL; we want engines to be "fast by default."

A glTF asset is composed of JSON describing the asset; binary .bin files containing geometry, animations, and skins; .glsl text files containing shaders; and image files for textures. Binary, glsl, and image files can also be embedded in the JSON.

glTF uses JSON because it is cross-platform, compact, readable, allows validation, and minifies and compresses well.

Binary data is little endian. Binary blobs allow efficient creation of GL buffers and textures since they require no additional parsing, except perhaps decompression. An asset can have any number of binary files for flexibility for a wide array of applications.



glTF Goals

- Balanced Feature Set

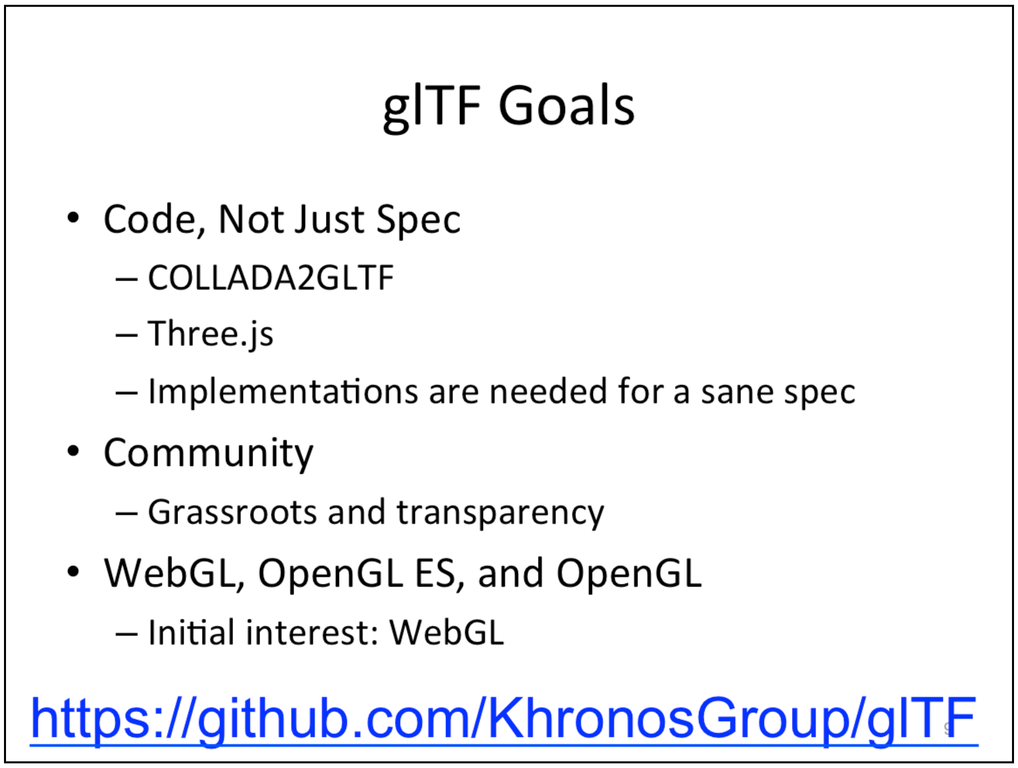
WebGL < glTF < COLLADA

- Extensible

8

glTF has more features than WebGL, like a node hierarchy, animation, and skins, but less features than an interchange format, like physics and spline representations. It tries to strike a balance between providing enough features and being bloated.

glTF is extensible by allowing extra JSON properties (literally named “extra : {}”) on each property, which is forward-compatible.



glTF Goals

- Code, Not Just Spec
  - COLLADA2GLTF
  - Three.js
  - Implementations are needed for a sane spec
- Community
  - Grassroots and transparency
- WebGL, OpenGL ES, and OpenGL
  - Initial interest: WebGL

https://github.com/KhronosGroup/glTF

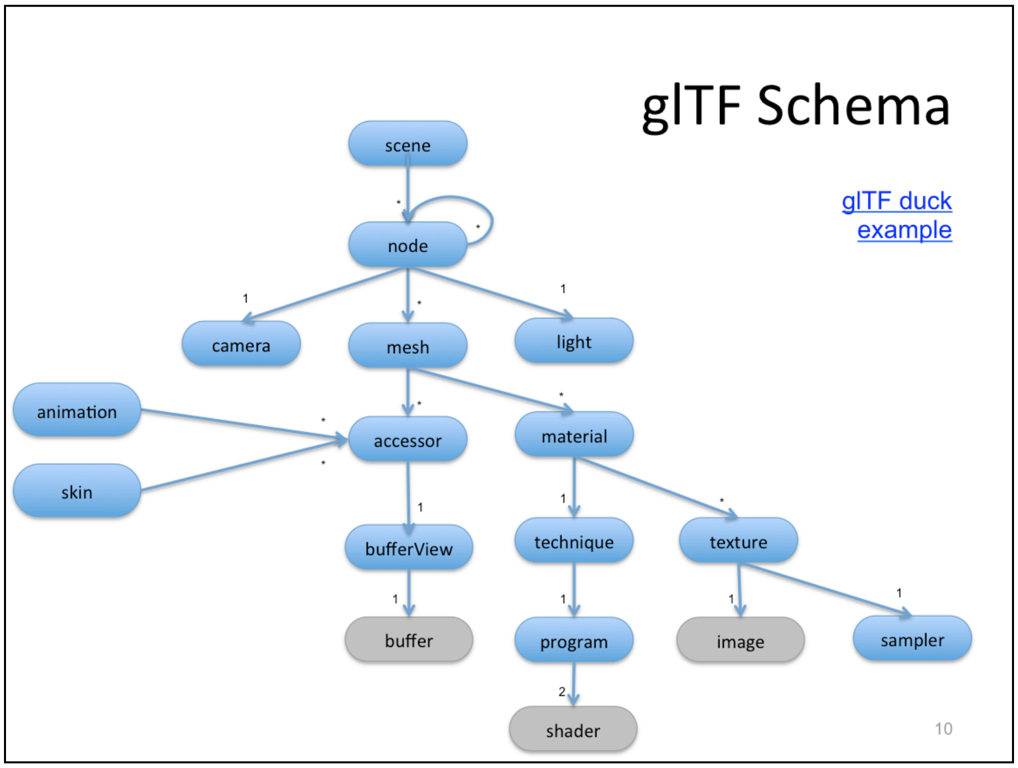
Like WebGL, glTF itself is really just a spec (and supporting schema). However, a spec alone is not useful to developers. There is an open-source COLLADA-to-glTF converter [1], which allows engine developers to save on the amount of code they have to write; in simple cases, we won't need any additional content pipeline code.

Three.js is widely used by the WebGL community so there is an open-source glTF loader for Three.js. We also believe that it's impossible to write a reasonable spec without implementing it so everyone working on the spec implemented a glTF loader, sometimes even more than one.

All of the spec work, and COLLADA2GLTF and Three.js loader code, was done in the open on github. This was useful for external bug fixes and more folks joining in on discussions.

glTF is designed for WebGL, OpenGL ES, and OpenGL, but initial adoption is primarily with WebGL since the engines are still emerging. Established engines like Unity and C4 already have a runtime format and have written supporting tools.

[1] https://github.com/KhronosGroup/glTF



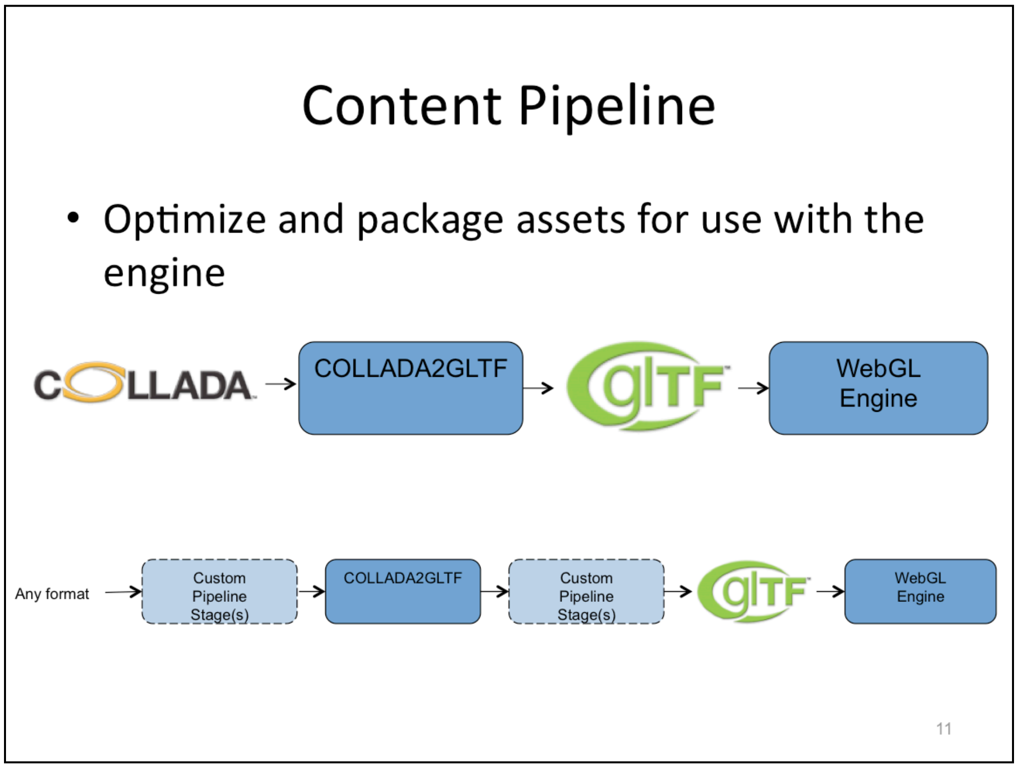
Here is the layout of a glTF asset. From the bottom up:

Geometry

- buffer – binary blob. Can be combination of geometry, animation, and skins
- bufferView – subset of buffer with target info (ARRAY_BUFFER, ELEMENT_BUFFER, animation/skin)
- accessor – subset of bufferView with type info, e.g., float-pointing. Similar to glVertexAttribPointer
    - For example, a bufferView may be all vertices in the asset (think glBufferData), where as an accessor may be an individual attribute for a mesh (think glVertexAttribPointer)
- mesh – (composed of primitives, not shown) – corresponds to glDrawElements
- node – one or more meshes, plus transform, plus references to child nodes

Material

- image – Image file
- sampler – texture filter and wrap modes, think glTexParameter
- texture – think glTexImage2D

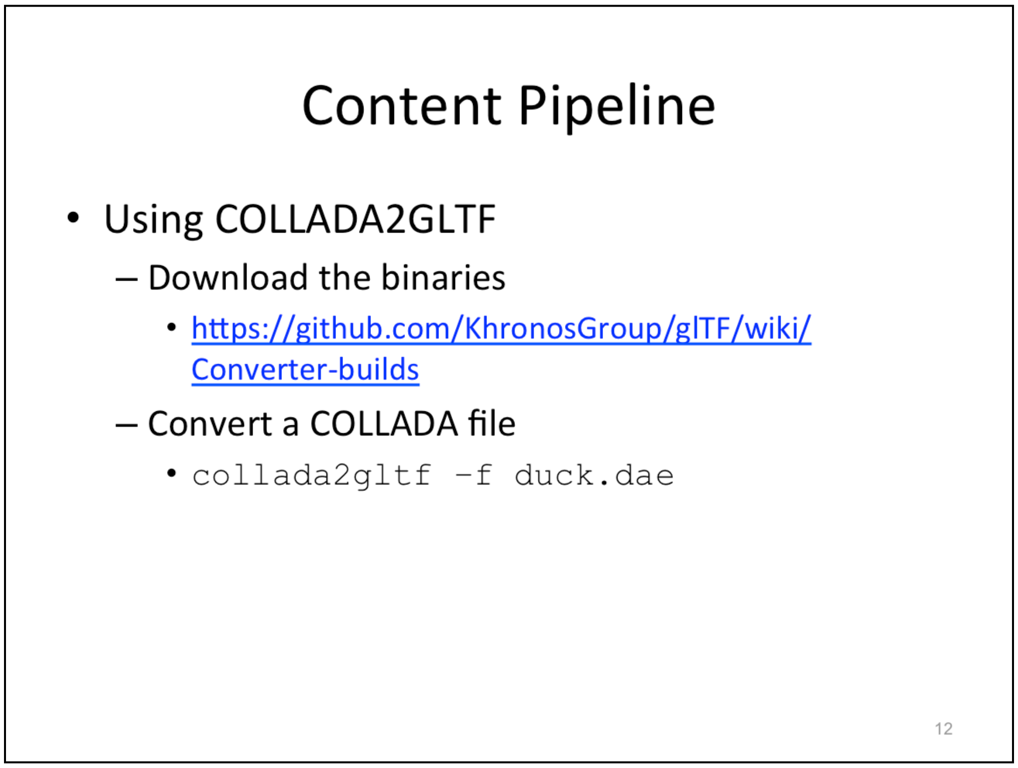


Content Pipeline

- Optimize and package assets for use with the engine

In general, the content pipeline is code outside the engine that processes and creates the final runtime asset. This is not usually all coded from scratch. Instead it is a combination of tools, often in different languages, from different third-parties, e.g., texture compression, mesh compression, etc.

For glTF, COLLADA2GLTF is the open-source COLLADA-to-glTF converter. As shown in the top pipeline, it can be used alone as a content pipeline. It can also be used with other tools to create a content pipeline. For example, in NORAD Tracks Santa, we added an optimization stage before COLLADA2GLTF to remove leaf nodes with just transforms that where included in the exported model.

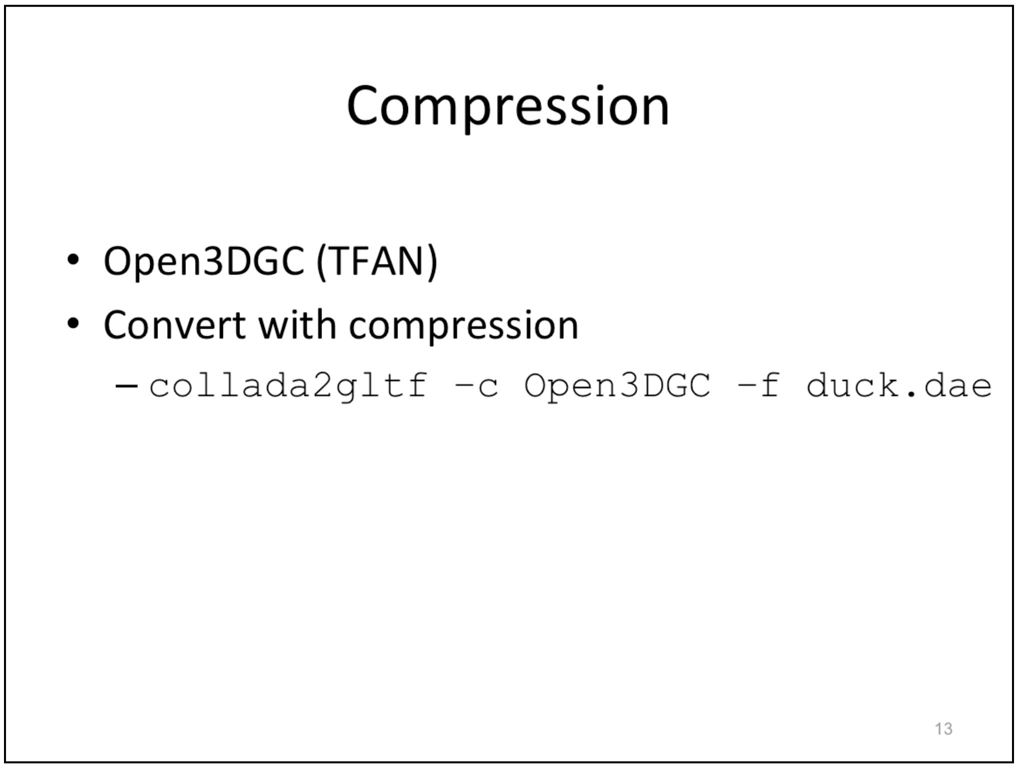


Content Pipeline

- Using COLLADA2GLTF
  - Download the binaries
    - https://github.com/KhronosGroup/glTF/wiki/Converter-builds
  - Convert a COLLADA file
    - `collada2gltf -f duck.dae`

12

Binaries for COLLADA2GLTF are on github. In the simplest case, we use it by passing the input COLLADA file with the –f argument, which then generates the glTF asset (.json file, .bin file, and .glsl files).



Compression

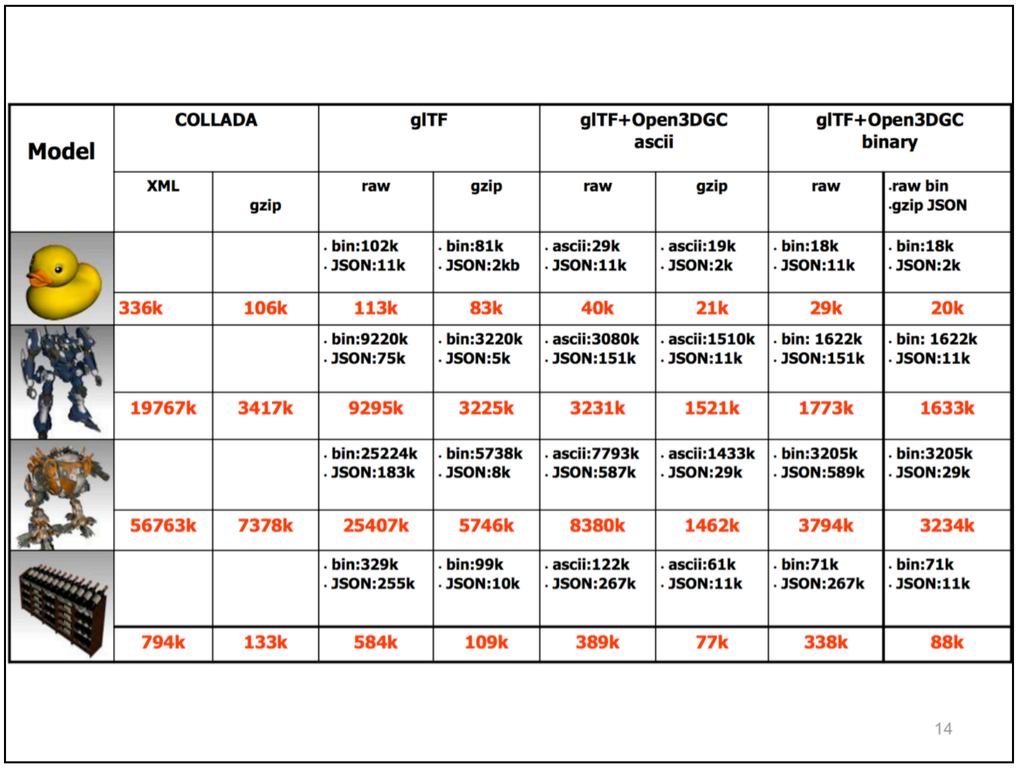
- Open3DGC (TFAN)
- Convert with compression
  - `collada2gltf -c Open3DGC -f duck.dae`

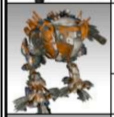
13

For WebGL, especially mobile, compression is important to reduce network bandwidth. Decompression needs to be fast enough so it doesn't drown out the bandwidth savings.

COLLADA2GLTF has optional Open3DGC compression [1], which is O(n) to decompress. Open3DGC is a TFAN compression that creates fans, quantizes, and uses parallelogram prediction.

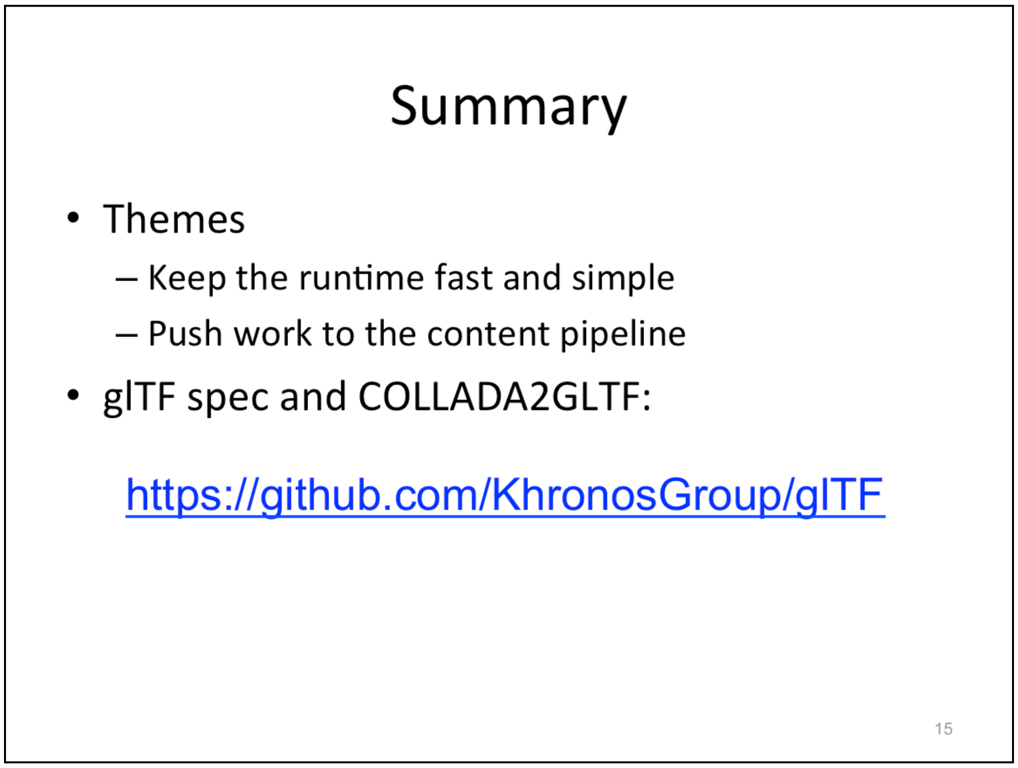
[1] http://kmamou.blogspot.com/2013/07/open-3d-graphics-compression-open3dgc.html



Model	COLLADA		glTF		glTF+Open3DGC ascii		glTF+Open3DGC binary	
	XML	gzip	raw	gzip	raw	gzip	raw	·raw bin ·gzip JSON
			· bin:102k · JSON:11k	· bin:81k · JSON:2kb	· ascii:29k · JSON:11k	· ascii:19k · JSON:2k	· bin:18k · JSON:11k	· bin:18k · JSON:2k
	336k	106k	113k	83k	40k	21k	29k	20k
			· bin:9220k · JSON:75k	· bin:3220k · JSON:5k	· ascii:3080k · JSON:151k	· ascii:1510k · JSON:11k	· bin: 1622k · JSON:151k	· bin: 1622k · JSON:11k
	19767k	3417k	9295k	3225k	3231k	1521k	1773k	1633k
			· bin:25224k · JSON:183k	· bin:5738k · JSON:8k	· ascii:7793k · JSON:587k	· ascii:1433k · JSON:29k	· bin:3205k · JSON:589k	· bin:3205k · JSON:29k
	56763k	7378k	25407k	5746k	8380k	1462k	3794k	3234k
			· bin:329k · JSON:255k	· bin:99k · JSON:10k	· ascii:122k · JSON:267k	· ascii:61k · JSON:11k	· bin:71k · JSON:267k	· bin:71k · JSON:11k
	794k	133k	584k	109k	389k	77k	338k	88k

14

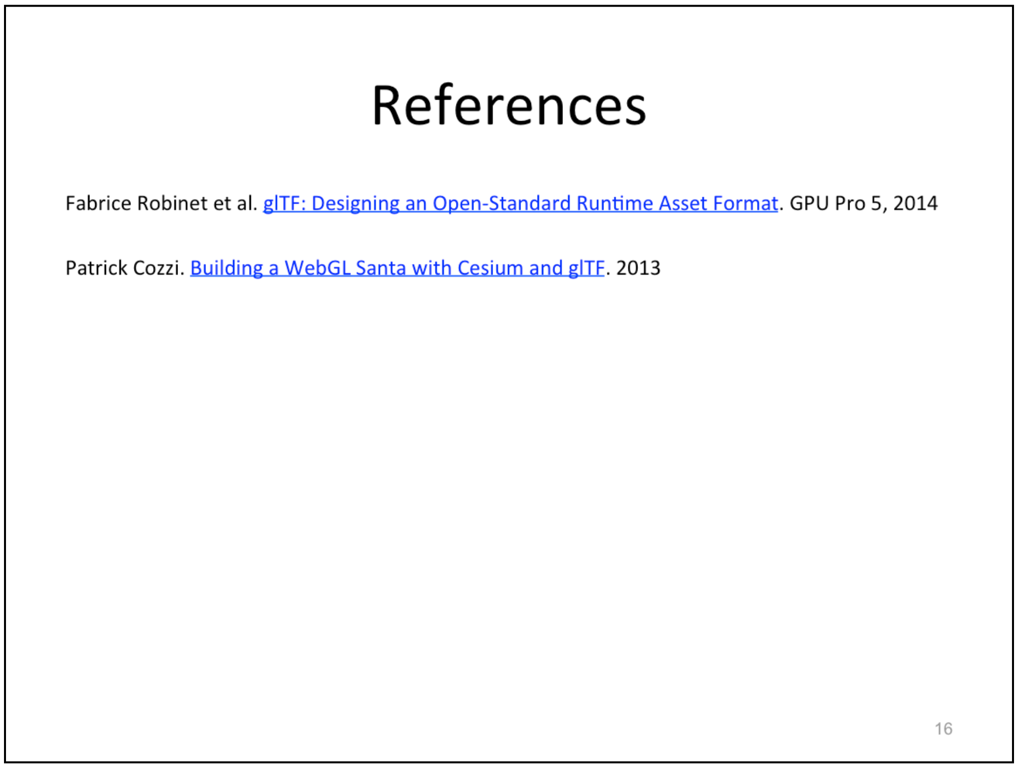
Here, the best results are the use Open3DGC ASCII (gzipped), which can be ~4x smaller than glTF alone (gzipped).


Summary

- Themes
  - Keep the runtime fast and simple
  - Push work to the content pipeline
- glTF spec and COLLADA2GLTF:

  https://github.com/KhronosGroup/glTF

15



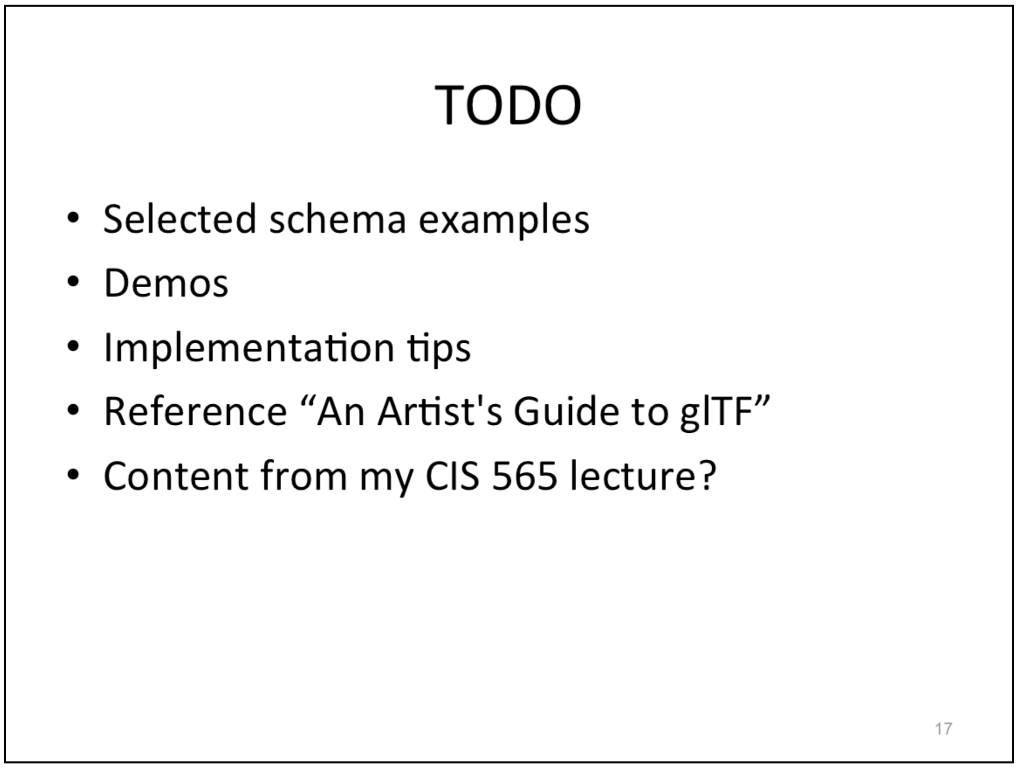
References

Fabrice Robinet et al. glTF: Designing an Open-Standard Runtime Asset Format. GPU Pro 5, 2014

Patrick Cozzi. Building a WebGL Santa with Cesium and glTF. 2013

16

Khronos logos from http://www.khronos.org/legal/trademarks/



TODO

- Selected schema examples
- Demos
- Implementation tips
- Reference “An Artist's Guide to glTF”
- Content from my CIS 565 lecture?

17