COMP3320 Introduction to OpenGL

Alex Biddulph

The University of Newcastle, Australia

Based on the work provided at www.learnopengl.com

Semester 2, 2021

The Open-Asset-Importer-Lib

- Asset importer library supporting multiple 3D model formats
- Provides some post-processing support
- API provided for C/C++
- Languange bindings available for C#, Java, Python, Delphi, D
- Can run on Android and iOS

Examples

Check out ☞ assimp.org for full details

Assimp Supported Formats

3D	**3DS**	3MF	AC
AC3D	ACC	AMJ	ASE
ASK	B3D	**BLEND**	BVH
CMS	COB	**DAE/Collada**	DXF
ENFF	FBX	glTF 1.0 + GLB	glTF 2.0
HMB	IFC-STEP	IRR / IRRMESH	LWO
LWS	LXO	MD2	MD3
MD5	MDC	MDL	MESH / MESH.XML
MOT	MS3D	NDO	NFF
OBJ	OFF	OGEX	PLY
PMX	PRJ	Q3O	Q3S
RAW	SCN	SIB	SMD
STP	**STL**	TER	UC
VTA	X	X3D	XGL
		ZGL	

Assimp Post Processing Support

- Normal generation
- Tangent generation
- Triangulation
- Removal of degenerate primitives
- Removal of duplicate vertices
- Index generation
- Lots more

Assimp Model Structure

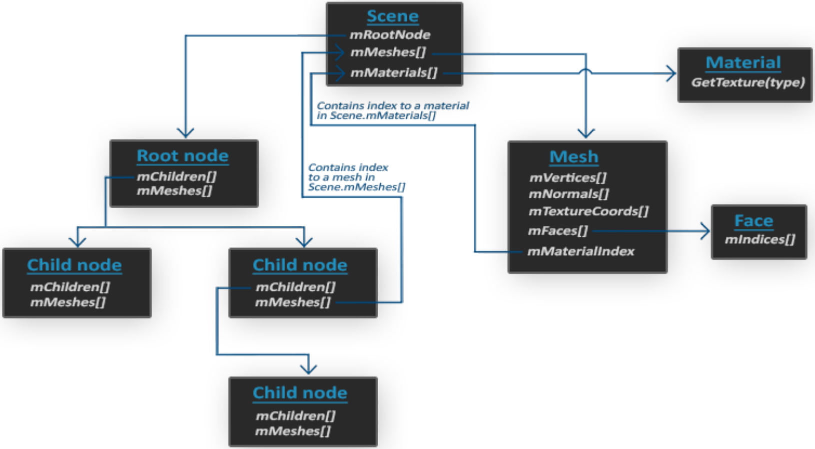

Figure: Image sourced from `learnopengl.com/Model-Loading/Assimp`

Assimp Model Structure

- ☞`Scene` is the root node of the imported model
- `Scene` contains all meshes and materials as well as a link to root node of the meshes
- `Root Node` starts a tree structure that contains links to meshes and child nodes
- Each ☞`Mesh` contains vertices, normals, texture coordinates, faces, and a link to materials (textures)
- Only vertices and faces is guaranteed to be in a `Mesh`, the rest are only there if they were in the model or you asked Assimp to calculate them for you
- Each ☞`Face` contains indices
- Each ☞`Material` contains information about a texture

Assimp and OpenGL

- Not geared at OpenGL
- Best to restructure the `Scene` data to make it easier to work with OpenGL
  - Create our own `Vertex` class which encapsulates position, normal, and texture coordinates
  - Create our own `Mesh` class which encapsulates VAOs, VBOs, EBOs, `Vertex` data, and textures
  - The `Mesh` class will also bind its own textures and render its own indices/vertices
  - The program's main render loop can now be reduced to calling `model.render()` for each loaded model and setting up uniforms for lighting

OpenAL

- Software interface for audio hardware
- Meant to resemble the OpenGL API
- A means to generate audio in a simulated 3D space
- OpenAL includes both the core API as well as OS bindings (unlike OpenGL)
- Can handle sound source directivity, distance-related attenuation, Doppler effects, and environmental effects
    - Reflection,
    - Obstruction,
    - Transmission, and
    - Reverberation

OpenAL Structure

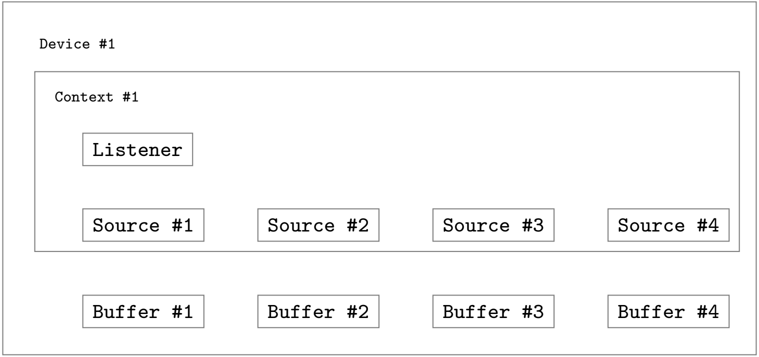

Figure: Image recreated from ☞ OpenAL Programmers Guide, Page 8

OpenAL Structure

- `Buffer`s are filled with audio data
  - Need to use an external library for this, similar to OpenGL and textures
  - ☞`libsndfile` is one option for this
  - ☞Supported formats and encodings can be seen here
- A `Buffer` is then attached to a `Source`
- There can be multiple `Source`s per context
- A `Source` has a position and an orientation (and other properties)
- The position and orientation of a `Source` relative to the `Listener` dictates how the `Source` is heard
- There can only be 1 `Listener` per context
- Update the positions, orientations, and velocities of the `Listener` and `Source`s dynamically to get convincing 3D audio effects

OpenAL Properties

- `Listener` properties
  - Gain
  - Position
  - Velocity
  - Orientation (`position` and `up`)
- `Source` properties
  - Pitch
  - Min gain, Gain, Max gain
  - Max distance, Reference distance
  - Rolloff factor
  - Position, velocity, direction
  - Source relative
  - Looping
  - .... many more

OpenAL Workflow

1. Determine which playback device you want to use via enumeration and open it
   - If there is only one playback device on your system you can use the default device
2. Create and open a context on the device
3. Set up initial listener properties
4. For each source
   1. Create source and set properties
   2. Load in audio data to a raw PCM format and transfer to OpenAL buffer
   3. Associate buffer with OpenAL source
5. Play sources when deemed appropriate
   - OpenAL will play sources asynchronously

More Information

Examples

See ☞OpenAL Programmers Guide for more details on using OpenAL

Examples

See ☞OpenAL short example for brief tutorial