

# GAM 695 Research

Tool Development with UI for C++  
Azul Converter

*By: Kerwin Ghigliotty Rivera*

*DepaulID: 1938268*

# Prototype Design

Initial prototype design was straight forward

Load a GLB File into the application using Open Dialog (before we were just using command line inputs which can be messy when there are options added)

The 3D model would be shown on screen and a few fields would populate.

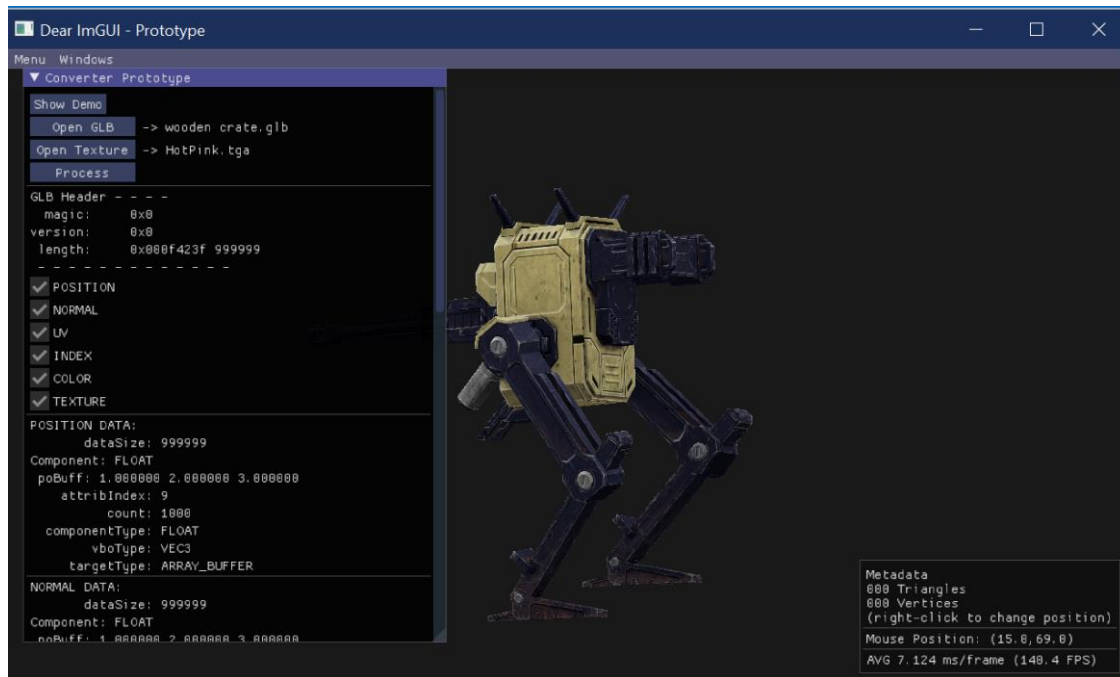
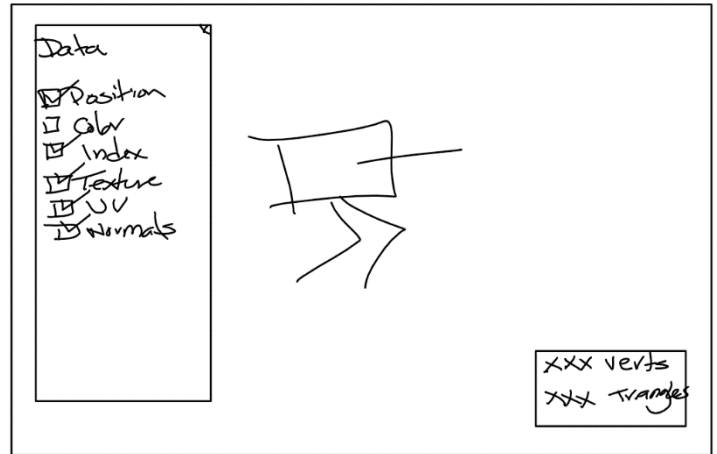
The metadata section would show the number of verts and triangles in the mesh (or total meshes)

The Data section would fill with the information regarding all of the vbos currently enabled in the model (Position, Normal, Index, Color, Texture, UV)

The user would then be able to directly affect the enabled property of each vbo and have it kept or removed from the model at the time of export.

Concept was created using Dear ImGui and at the moment it called for a few other changes

## Concept of Prototype V0.1

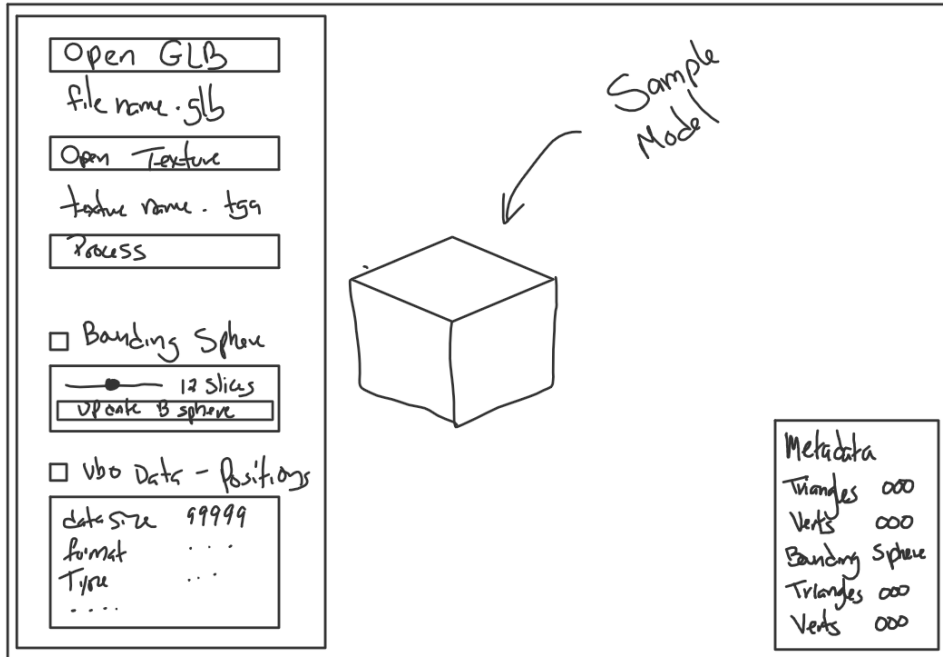


Prototype V0.1 created using Dear ImGui using dummy data

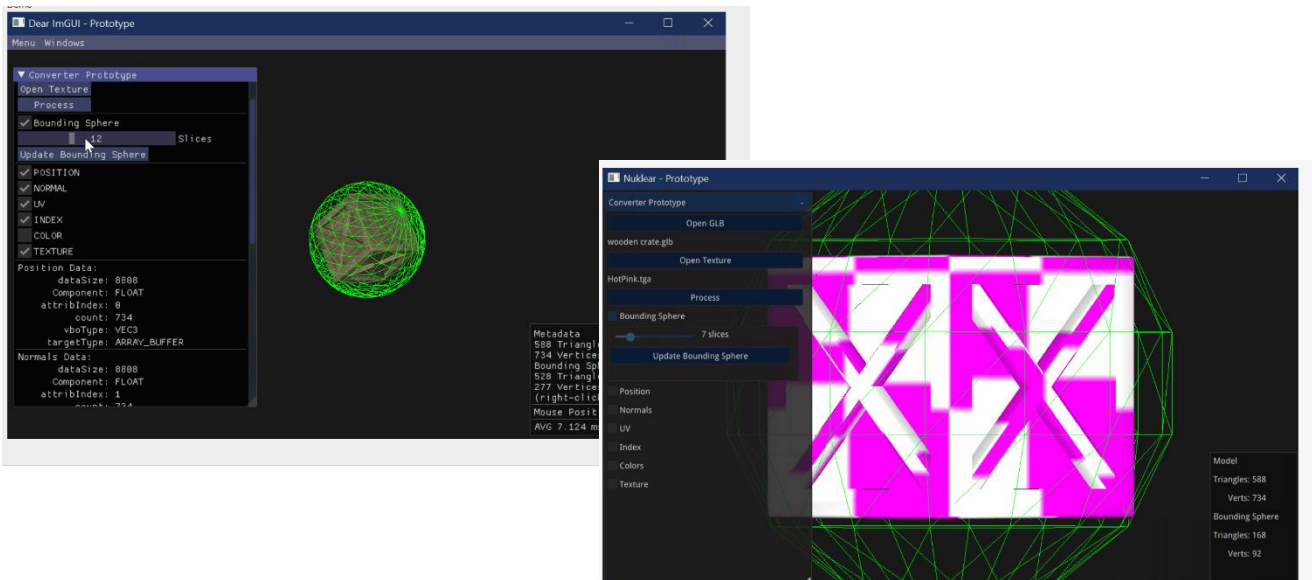
Some data from the GLB file was unnecessary, like GLB header checks and so those were removed. Additionally, a Bounding Sphere section was added, the idea would be that the user could see the bounding sphere and modify the number of slices in the bounding sphere, reducing its cost to generate at the cost of lower accuracy.

The Bounding Sphere properties would also be shown in the Metadata section at all times.

And with that along with some other engine changes prototype V0.2 concept was created.



Prototype V0.2 created in both Dear ImGui and Nuklear



## UI Libraries for C++

Dear ImGui

Link: <https://github.com/ocornut/imgui>

Developed and maintained by Github user ocornut (Omar Cornut) and a lot more other users, licensed under the MIT License

Bloat-Free graphical user interface library for C++. Renders Vertex Buffers that we can implement into our 3D engine, this requires a back end to interact with the elements. It is designed to be used for tool development (not for average end-user consumption). Used and supported by many big game studios.

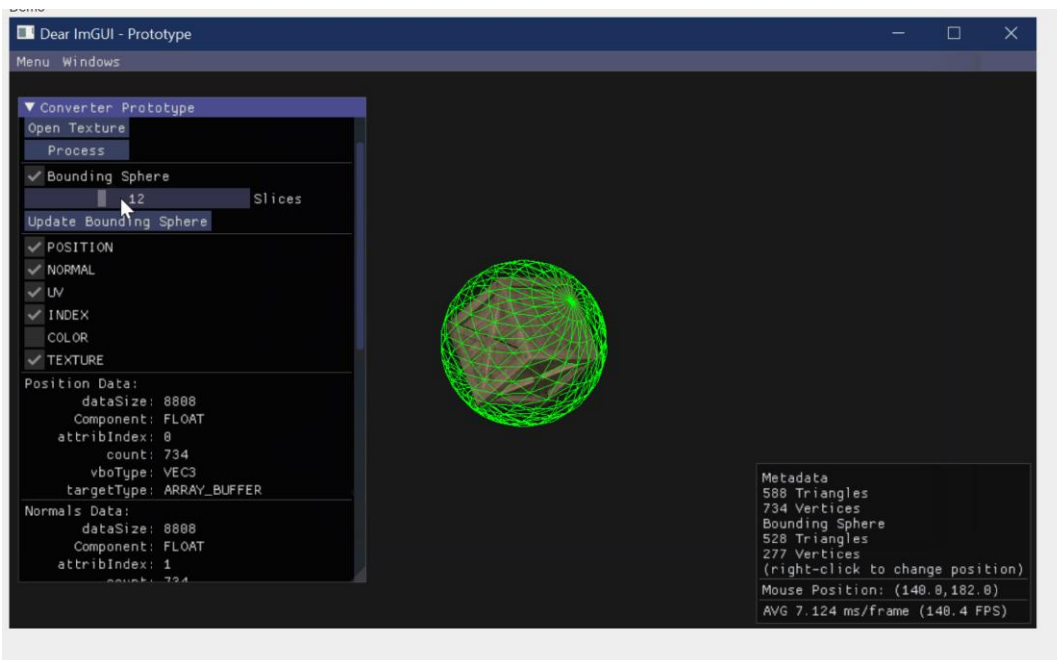
This is done in an Immediate Rendering paradigm which basically means that a line of code is translated directly into a feature of the GUI

For example, the code on the right creates a menu drop down with "Open GLB File" and "Save" buttons.

Every time this code is read the menu is created on screen, meaning that for this menu to persist on the GUI interface it needs to be executed in a loop.

This allows for fast development, but potentially costly if done poorly.

```
if(ImGui::BeginMenu("Menu"))
{
    if (ImGui::MenuItem("Open GLB File", "Ctrl+O"))
    {
        Trace::out("File Dialog Open\n");
    }
    if (ImGui::MenuItem("Save", "Ctrl+S"))
    {
        Trace::out("File Dialog Save\n");
    }
}
ImGui::EndMenu();
```



Prototype V0.2 Created in Dear ImGui

## *PROS*

Setting it up was super simple, after importing the main files and the appropriate renderer file creating UI elements was very easy.

A lot of functionality already developed makes it easy to just pick from an example and replicate

A lot more support compared to Nuklear, with ImGui having 5940 forks with 973 watchers at the time of this documentation

## *CONS*

The framework conflicts with our Memory Tracker due to it redefining placement new, turning Memory Tracker off makes this work.

Need to be very aware of how the GUI is implemented, creating a window frame without closing is costly but this is a problem with all immediate rendering libraries.

## Nuklear

Link: <https://github.com/Immediate-Mode-UI/Nuklear>

Developed by Micha Mettke (vurtun on GitHub) as well as a 129+ contributors on GitHub

“It is a minimal-state, immediate-mode graphical user interface toolkit written in ANSI C and licensed under public domain. It is designed as a simple embeddable user interface for applications and does not have any dependencies, a default render backend or OS window/input handling but instead provides a highly modular, library-based approach with simple input state for input and draw commands describing primitive shapes as output. So instead of providing a layered library that tries to abstract over a number of platform and render backends, it focuses only on the actual UI.” – Taken from their GitHub project Readme.md

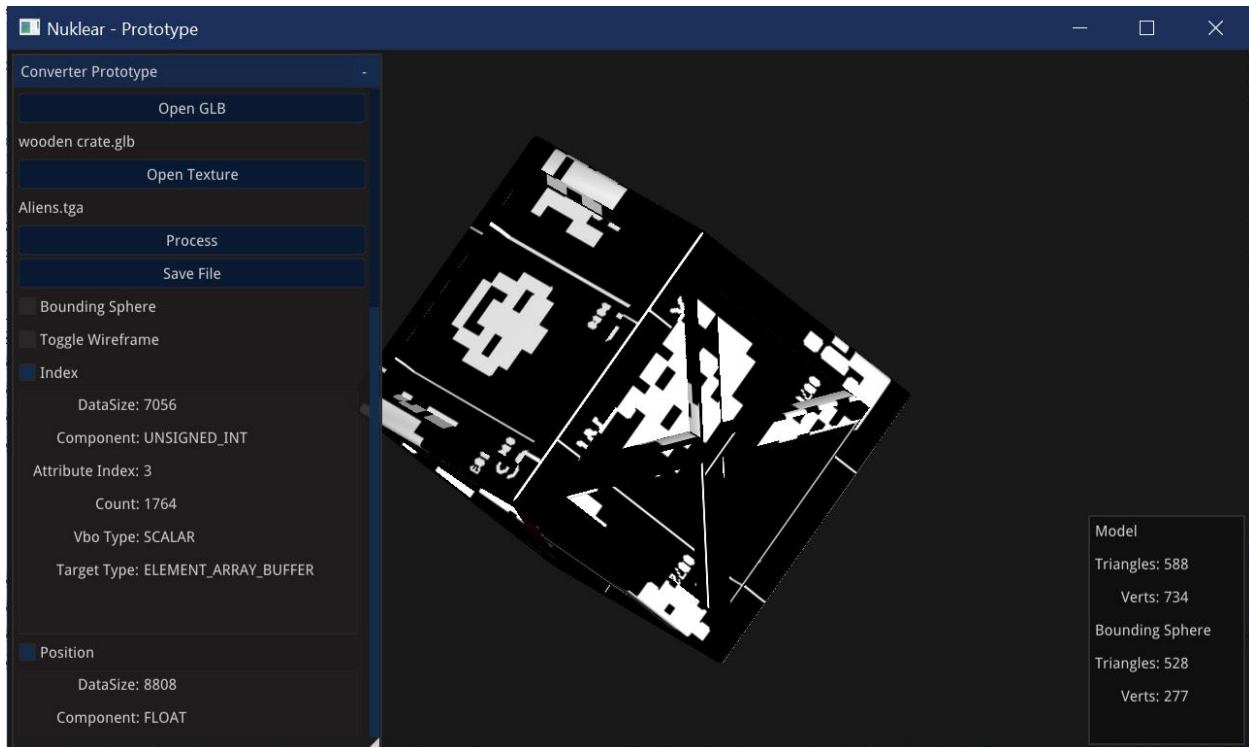
Uses the same Immediate paradigm as DearImGui but this time it only uses a single header file with some support files.

Prototyping the Azul Converter on Nuklear at first was a bit of a problem, there is less functionality out of the box in Nuklear as opposed to DearImGui

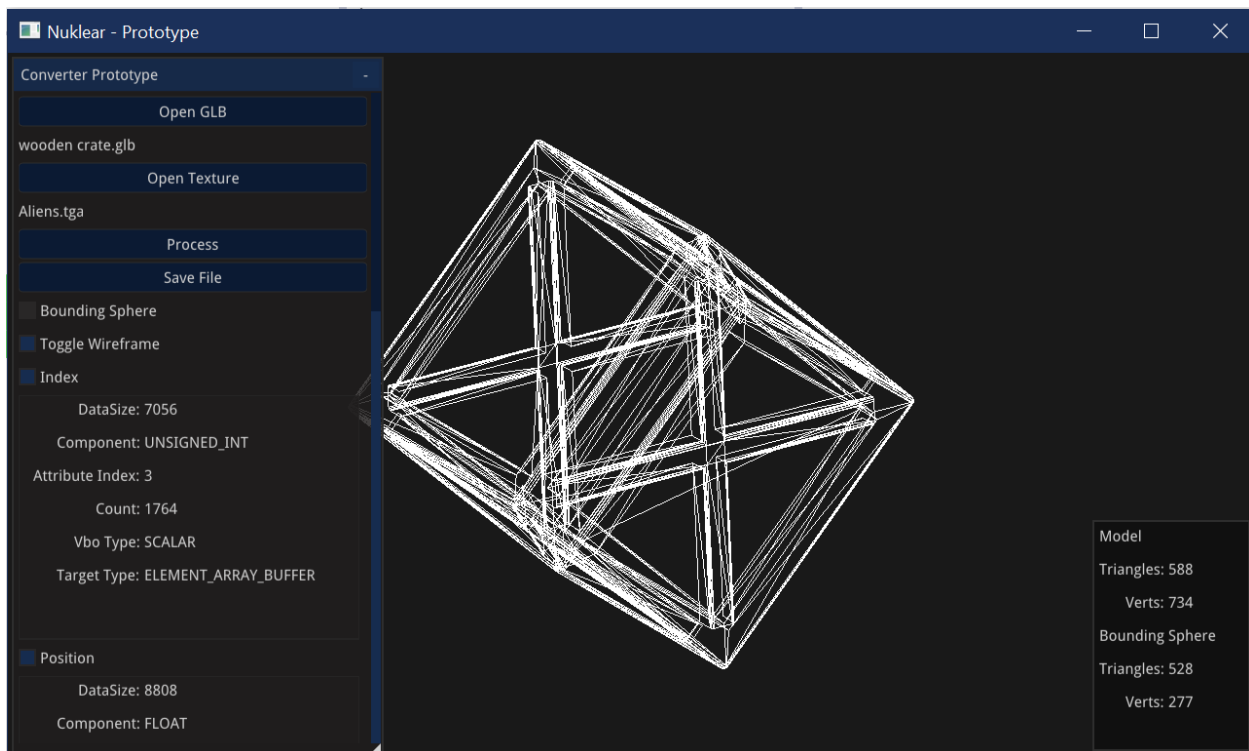
ImGui has a lot of already established functionality whereas in Nuklear a lot of functionality that you want you have to dig up and modify yourself (Transparency on the bottom right window is an example of this). Which gives you a sense of ownership when you crack it and make it work.

```
nk_style* transparentStyle = &context->ctx->style;
nk_style_push_color(context->ctx, &transparentStyle->window.background, nk_rgba(0,0,0,89));
nk_style_push_style_item(context->ctx, &transparentStyle->window.fixed_background, nk_style_item_color(nk_rgba(0,0,0,89)));
if (nk_begin(context->ctx, "Metadata", nk_rect((float)(context->windowWidth - 165),
                                             (float)(context->windowHeight - 240), 160, 235),
      NK_WINDOW_BORDER | NK_WINDOW_NO_INPUT))
{
    unsigned int count = 0;
    nk_layout_row_dynamic(context->ctx, 30, 1);
    nk_label(context->ctx, "Model", NK_TEXT_ALIGN_LEFT);
    nk_layout_row_dynamic(context->ctx, 30, 2);
    nk_label(context->ctx, "Triangles:", NK_TEXT_ALIGN_RIGHT);
    count = context->mA == nullptr ? 0 : context->mA->aMeshData[1].triCount;
    nk_label(context->ctx, std::to_string(count).c_str(), NK_TEXT_ALIGN_LEFT);
    nk_label(context->ctx, "Verts:", NK_TEXT_ALIGN_RIGHT);
    count = context->mA == nullptr ? 0 : context->mA->aMeshData[1].vertCount;
    nk_label(context->ctx, std::to_string(count).c_str(), NK_TEXT_ALIGN_LEFT);

    nk_layout_row_dynamic(context->ctx, 30, 1);
    nk_label(context->ctx, "Bounding Sphere", NK_TEXT_ALIGN_LEFT);
    nk_layout_row_dynamic(context->ctx, 30, 2);
    nk_label(context->ctx, "Triangles:", NK_TEXT_ALIGN_RIGHT);
    count = context->mA == nullptr ? 0 : context->mA->aMeshData[0].triCount;
    nk_label(context->ctx, std::to_string(count).c_str(), NK_TEXT_ALIGN_LEFT);
    nk_label(context->ctx, "Verts:", NK_TEXT_ALIGN_RIGHT);
    count = context->mA == nullptr ? 0 : context->mA->aMeshData[0].vertCount;
    nk_label(context->ctx, std::to_string(count).c_str(), NK_TEXT_ALIGN_LEFT);
}
nk_end(context->ctx);
```



*Prototype V0.3 Created in Nuklear*



*Wireframe toggle test on Prototype V0.3 with Nuklear*

### *Pros*

Simple setup since its only one file + supporting files you may need + renderer file

Some demos show functionality you can plug and play easily

### *Cons*

Conflicts with our memory tracker since it is also redefining placement new, turning Memory Tracker off makes this work.

Commits to Nuklear are less often than DearImGui as Nuklear has less support (1111 forks with 567 watchers).







A lot of functionality you must dig through code and reverse engineer as no real examples are given.

Author does not have time to keep updating it since it is open source, and it is not his priority.



## Results from Final Prototype

Using the latest version of my tool I am able to strip off the UV, Normal and Texture vbos and these are my results (for the “wooden crate.glb” file)

 wooden crate_3_Slice_ALL.mt.proto.azul	3/8/2022 11:29 AM	AZUL File	4,128 KB
 wooden crate_3_Slice_NT_NUV_NN.m.proto.azul	3/8/2022 11:34 AM	AZUL File	17 KB
 wooden crate_12_Slice_ALL.mt.proto.azul	3/8/2022 11:29 AM	AZUL File	4,140 KB
 wooden crate_12_Slice_NT_NUV_NN.m.proto.azul	3/8/2022 11:31 AM	AZUL File	29 KB
 wooden crate_24_Slice_ALL.mt.proto.azul	3/8/2022 11:30 AM	AZUL File	4,179 KB
 wooden crate_24_Slice_NT_NUV_NN.m.proto.azul	3/8/2022 11:30 AM	AZUL File	69 KB

The ratio is consistent since from 3 -> 12 slice it is only a 12KB ish increase in size and from 12 -> 24 is a 40KB ish increase in size

Minus the data from Texture, Normal and UV VBOS 4110KB (bounding sphere makes up for the extra KB in the 3 and 12 slice variants)

Here are the models working in my engine from last quarter

Top are Models with 3 Slice Bounding Sphere with and without Texture, Normal and UV information

Middle are Models with 12 Slice Bounding Sphere with and without Texture, Normal and UV information

Bottom are Models with 24 Slice Bounding Sphere with and without Texture, Normal and UV information

