My Journey with 3DS Programming

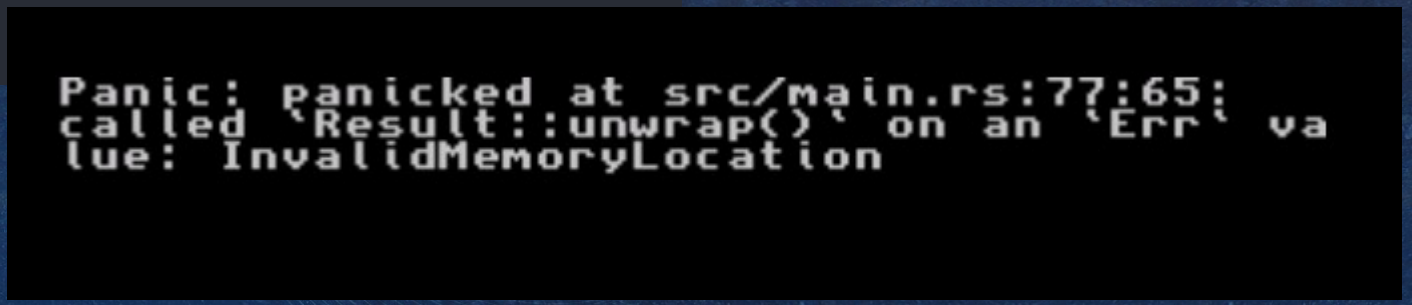
Getting Started

- DevKitPro - https://devkitpro.org/
- Already great tools
- Uses a custom pacman-based manager for installing the different tools
- Libctru and libcitro3d C libraries
- Rust3ds org - https://github.com/rust3ds
- rust3ds/ctru-rs wrapper is already quite good
- Cargo 3DS wrapper for cargo to make compiling, packaging, and even remotely running on the 3DS super easy (using 3dslink)

Using ctru-rs

- Can access most of the hardware
- Tools for easy console output (and with Rust, easy debugging!)

```rust
fn main() {
    let gfx: Gfx = Gfx::new().unwrap();
    let apt: Apt = Apt::new().unwrap();
    let mut hid: Hid = Hid::new().unwrap();
    let mut citro: Instance = citro3d::Instance::new().unwrap();

    let _console: Console<'_> = Console::new(screen: gfx.bottom_screen.borrow_mut());

    std::panic::set_hook(Box::new(|info: &PanicHookInfo<'_>| {
        println!("Panic: {info}");

        unsafe {
            loop {
                ctru_sys::hidScanInput();
                let keys: u32 = ctru_sys::hidKeysDown();
                if KeyPad::from_bits_truncate(bits: keys).contains(KeyPad::START) {
                    break;
                }
            }
        }
    }));
```



Using ctru-rs

- Directly manipulate the framebuffer
- Is slow :(
- But there is a GPU we can use instead

3DS GPU - PICA200

- Developed by Digital Media Professionals Inc (a Japanese GPU design startup)
- Used for embedded devices
- Licensed by Nintendo for the 3DS
- Programmable Vertex and Geometry shaders
- No Fragment shader
  - Uses a fixed-function “TexEnv” and “LightEnv” pipeline for fragment operations

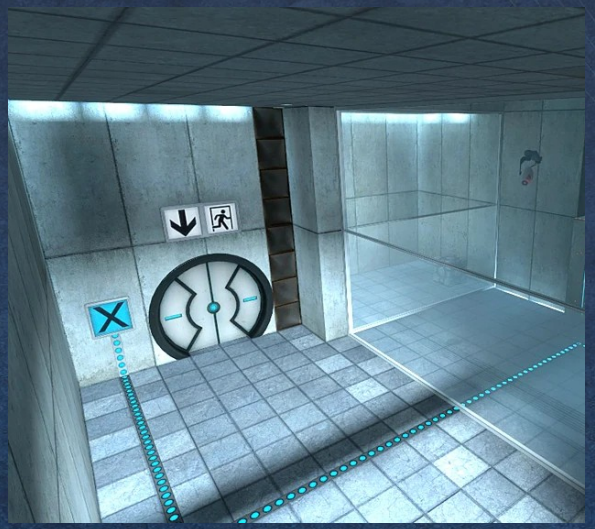
Tangent: TexEnvs

- No fragment shader
- Uses up to 6 stages of a “Texture Environment”
- Each stage takes up to 3 sources (vertex shader outputs, textures, output from previous stage)
- Performs a configurable set of operations on them
- Each stage is also split into RGB and Alpha, which can be configured together or separately

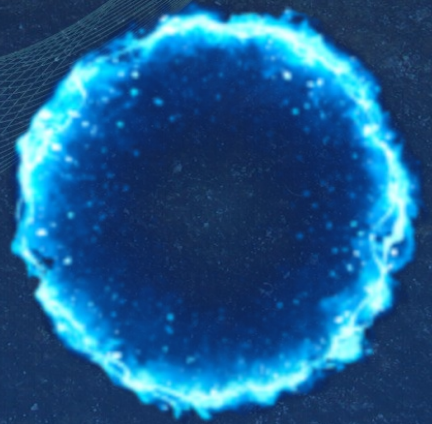
```
let textured_stage: TexEnv = texenv::TexEnv::new() TexEnv
    .sources(
        mode: texenv::Mode::BOTH,
        source0: texenv::Source::Texture0,
        source1: None,
        source2: None
    );
```

Tangent: TexEnvs – Portal effect

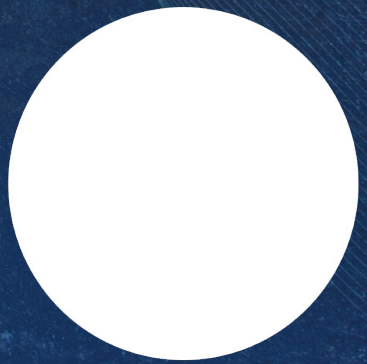
Texture0 – Scene (Rendered to texture)



Texture1 - Portal



Texture2 - Stencil

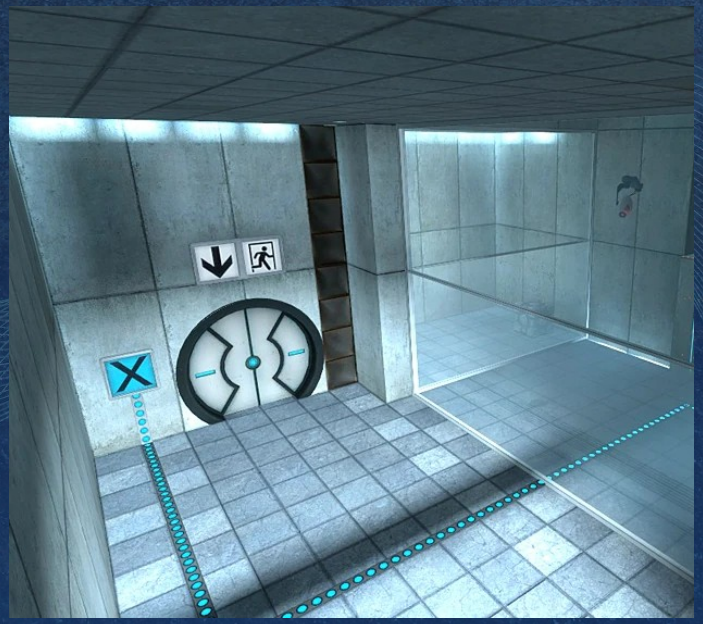


```
let stage0: TexEnv = texenv::TexEnv::new() TexEnv
    .sources(
        mode: texenv::Mode::BOTH,
        source0: texenv::Source::Texture0,
        source1: Some(texenv::Source::Texture2),
        source2: None
    ) TexEnv
    .op_rgb(
        o1: texenv::RGBOp::SrcColor,
        o2: Some(texenv::RGBOp::SrcAlpha),
        o3: None
    ) TexEnv
    .op_alpha(
        o1: texenv::RGBOp::SrcAlpha,
        o2: Some(texenv::RGBOp::SrcAlpha),
        o3: None
    ) TexEnv
    .func(mode: texenv::Mode::BOTH, func: texenv::CombineFunc::Modulate);

let stage1: TexEnv = texenv::TexEnv::new() TexEnv
    .sources(
        mode: texenv::Mode::BOTH,
        source0: texenv::Source::Previous,
        source1: Some(texenv::Source::Texture1),
        source2: Some(texenv::Source::Texture1)
    ) TexEnv
    .op_rgb(
        o1: texenv::RGBOp::SrcColor,
        o2: Some(texenv::RGBOp::SrcColor),
        o3: Some(texenv::RGBOp::OneMinusSrcAlpha)
    ) TexEnv
    .func(mode: texenv::Mode::RGB, func: texenv::CombineFunc::Interpolate) TexEnv
    .op_alpha(
        o1: texenv::AlphaOp::SrcAlpha,
        o2: Some(texenv::AlphaOp::SrcAlpha),
        o3: None
    ) TexEnv
    .func(mode: texenv::Mode::ALPHA, func: texenv::CombineFunc::Add);
```

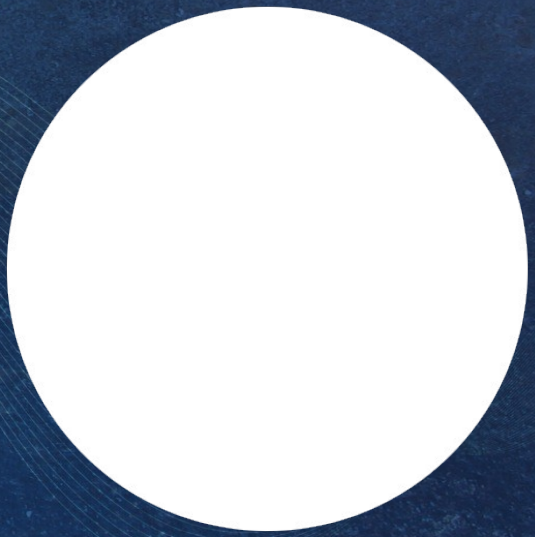
Tangent: TexEnvs – Portal effect

- Modulate scene texture colours/alpha by the stencil
- Modulate operation takes an input texture to modulate, and a texture to modulate by

```
let stage0: TexEnv = texenv::TexEnv::new() TexEnv
    .sources(
        mode: texenv::Mode::BOTH,
        source0: texenv::Source::Texture0,
        source1: Some(texenv::Source::Texture2),
        source2: None
    ) TexEnv
    .op_rgb(
        o1: texenv::RGBOp::SrcColor,
        o2: Some(texenv::RGBOp::SrcAlpha),
        o3: None
    ) TexEnv
    .op_alpha(
        o1: texenv::RGBOp::SrcAlpha,
        o2: Some(texenv::RGBOp::SrcAlpha),
        o3: None
    ) TexEnv
    .func(mode: texenv::Mode::BOTH, func: texenv::CombineFunc::Modulate);
```



+

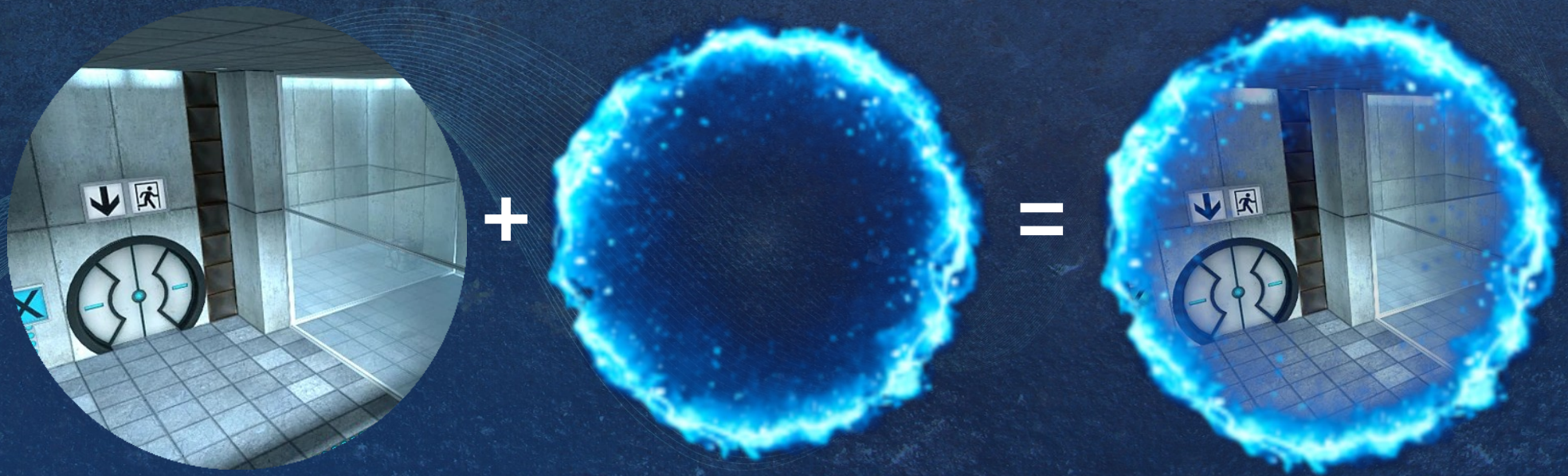


=

Tangent: TexEnvs – Portal effect

- Second stage combines the 2 textures
- RGB: Take the colours of both textures and interpolate between them by the alpha of the portal texture
- ALPHA: Subtract them

```
let stage1: TexEnv = texenv::TexEnv::new() TexEnv
    .sources(
        mode: texenv::Mode::BOTH,
        source0: texenv::Source::Previous,
        source1: Some(texenv::Source::Texture1),
        source2: Some(texenv::Source::Texture1)
    ) TexEnv
    .op_rgb(
        o1: texenv::RGBOp::SrcColor,
        o2: Some(texenv::RGBOp::SrcColor),
        o3: Some(texenv::RGBOp::OneMinusSrcAlpha)
    ) TexEnv
    .func(mode: texenv::Mode::RGB, func: texenv::CombineFunc::Interpolate) TexEnv
    .op_alpha(
        o1: texenv::AlphaOp::SrcAlpha,
        o2: Some(texenv::AlphaOp::SrcAlpha),
        o3: None
    ) TexEnv
    .func(mode: texenv::Mode::ALPHA, func: texenv::CombineFunc Subtract
```



Tangent: TexEnvs – Portal effect

- Or approximately…
- I haven't actually tested this
- But it's just to give an idea of the kind of vibe of what TexEnvs do

```
let stage0: TexEnv = texenv::TexEnv::new() TexEnv
    .sources(
        mode: texenv::Mode::BOTH,
        source0: texenv::Source::Texture0,
        source1: Some(texenv::Source::Texture2),
        source2: None
    ) TexEnv
    .op_rgb(
        o1: texenv::RGBOp::SrcColor,
        o2: Some(texenv::RGBOp::SrcAlpha),
        o3: None
    ) TexEnv
    .op_alpha(
        o1: texenv::RGBOp::SrcAlpha,
        o2: Some(texenv::RGBOp::SrcAlpha),
        o3: None
    ) TexEnv
    .func(mode: texenv::Mode::BOTH, func: texenv::CombineFunc::Modulate);

let stage1: TexEnv = texenv::TexEnv::new() TexEnv
    .sources(
        mode: texenv::Mode::BOTH,
        source0: texenv::Source::Previous,
        source1: Some(texenv::Source::Texture1),
        source2: Some(texenv::Source::Texture1)
    ) TexEnv
    .op_rgb(
        o1: texenv::RGBOp::SrcColor,
        o2: Some(texenv::RGBOp::SrcColor),
        o3: Some(texenv::RGBOp::OneMinusSrcAlpha)
    ) TexEnv
    .func(mode: texenv::Mode::RGB, func: texenv::CombineFunc::Interpolate) TexEnv
    .op_alpha(
        o1: texenv::AlphaOp::SrcAlpha,
        o2: Some(texenv::AlphaOp::SrcAlpha),
        o3: None
    ) TexEnv
    .func(mode: texenv::Mode::ALPHA, func: texenv::CombineFunc::Add);
```

Programming the PICA200

- Citro3d – C driver for the PICA200
- Picasso – Custom shader language and compiler
  - Assembly, not a high-level language
  - Operates entirely on vector registers (mostly 4 x float vectors)
  - Will go into more detail later

Registers [edit]

Name	Format	Type	Access	Written by	Description
v0-v15	vector	float	Read only	Application/Vertex-stream	Input registers.
o0-o15	vector	float	Write only	Vertex shader	Output registers.
r0-r15	vector	float	Read/Write	Vertex shader	Temporary registers.
c0-c95	vector	float	Read only	Application/Vertex-stream	Floating-point Constant registers.
i0-i3	vector	integer	Read only	Application	Integer Constant registers. (special purpose)
b0-b15	scalar	boolean	Read only	Application	Boolean Constant registers. (special purpose)
a0.x & a0.y	scalar	integer	Use/Write	Vertex shader	Address registers.
aL	scalar	integer	Use	Vertex shader	Loop count register.

Rust support for Citro3D?

- Citro3d – C driver for the PICA200
- Rust wrapper is not good…
- Lacks basic features and basically only supports loading vertices and rendering them to the screen
- Unsafe
  - Citro3D calls do not immediately have an effect, but queue an operation
  - Calling endFrame() dispatches the queue and performs all the operations
  - Meaning all resources used in any of those calls must live until after endFrame() completes
  - The current Rust wrapper does not ensure that, even though Rust has a powerful lifetime system..

Improving the wrapper

- Overhaul the API
- Have a “Frame” struct who’s lifetime represents the beginning and end of a frame (it calls endFrame() when it is dropped)
- Frame borrows all resources used in graphics calls, ensuring they live until the end of the frame
- What about following frames? State can hang around and continue to be used in later frames
- Do those resources need to be borrowed permanently..?
- No
- Have a “RenderPass” struct to which you must provide ALL the required state for a frame to be rendered
- If it is not provided to the RenderPass, it will not be used in the frame, and so does not need to be borrowed
- Also implement and merge a bunch more of the features other people had started, like textures, rendering to textures, indices, lighting envs, etc

Improving the wrapper

- RenderPass is provided with at minimum the shader program and vertex buffer + attribute info
- Can provide TexEnvs, LightEnvs, indices, textures, uniforms, etc
- Anything not provided will be given a default, so leftover state from previous frames is not reused by accident (e.g. a texture that has since been freed)

```
        // Render
        citro.render_frame_with(|frame: &mut Frame<'_, '_>| {
            screen_target.clear(flags: ClearFlags::ALL, rgba_color: CLEAR_COL, depth: 0);

            let body_pass: RenderPass<'_, '_, ScreenTarget<'_>, …> = RenderPass::new(
                &program,
                &screen_target,
                vbo_data: model1_slice,
                attribute_info: &attr_info
            ) RenderPass<'_, '_, ScreenTarget<'_>, …>
                .with_texenv_stages(texenvs: [&textured_stage]) RenderPass<'_, '_, ScreenTarget<'_>, …>
                .with_indices(&model1_inds) RenderPass<'_, '_, ScreenTarget<'_>, …>
                .with_texture(texture_unit: texture::TexUnit::TexUnit0, texture: &texture1) RenderPass<'_, '_, ScreenTarget<'_>, …>
                .with_vertex_uniforms([(uniform_proj, (mvp).into())]);
            frame.draw(&body_pass).unwrap();

            let wings_pass: RenderPass<'_, '_, ScreenTarget<'_>, …> = body_pass RenderPass<'_, '_, ScreenTarget<'_>, …>
                .with_vbo(vbo_data: model2_slice, attribute_info: &attr_info) RenderPass<'_, '_, ScreenTarget<'_>, …>
                .with_indices(&model2_inds) RenderPass<'_, '_, ScreenTarget<'_>, …>
                .with_texture(texture_unit: texture::TexUnit::TexUnit0, texture: &texture2);
            frame.draw(&wings_pass).unwrap();
        });

        total_frame_time = unsafe { ctru_sys::osGetTime() - frame_start_time };
    }
```

Improving the wrapper

- Solves all those lifetimes issues mentioned before
- Full configuration of the GPU state in one place makes it easy for the programmer to tell what is happening
- Can allow for more precise error-checking
  - e.g. If textures are referenced in the TexEnv, but no textures were provided, it can provide an informative error and avoid doing the pass and referencing an invalid texture

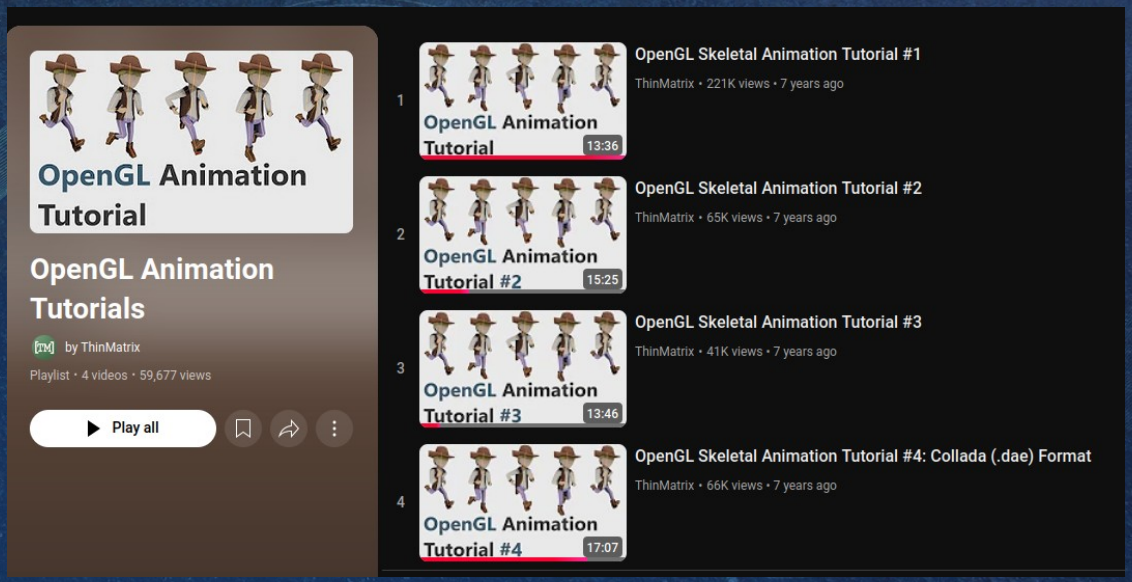
BUT

- Subsequent render calls or even entire frames may not need that much reconfiguration, so there is overhead

```
Hello, World!

  Frame time: 0ms
Panic: panicked at src/main.rs:196:36:
called `Result::unwrap()` on an `Err` va
lue: MissingTexture(TexUnit0)
```

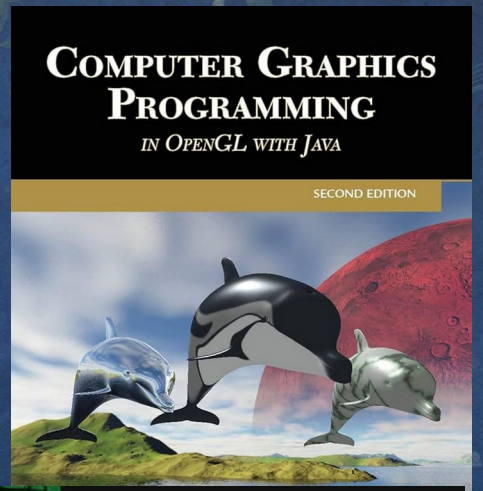
Skeletal Animation

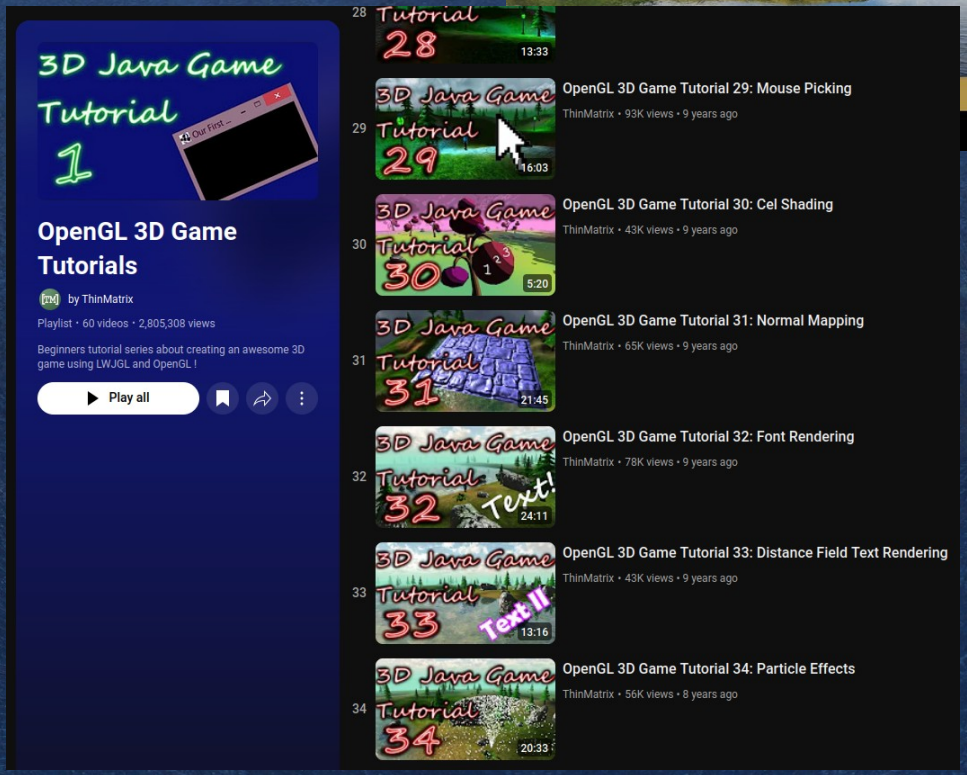
- At this point, I had formed a major goal to have a model exported from Blender rendering and animated
- Would require
  - Skeletal animation system
  - Parsing mesh, texture, skeleton, and animation data from an exported file
  - An animated Model in Blender
  - Rendering the parsed and animated data
- Implementation of the skeletal animation system heavily referenced a tutorial series by ThinMatrix



Tangent: How I got into 3D Graphics Programming

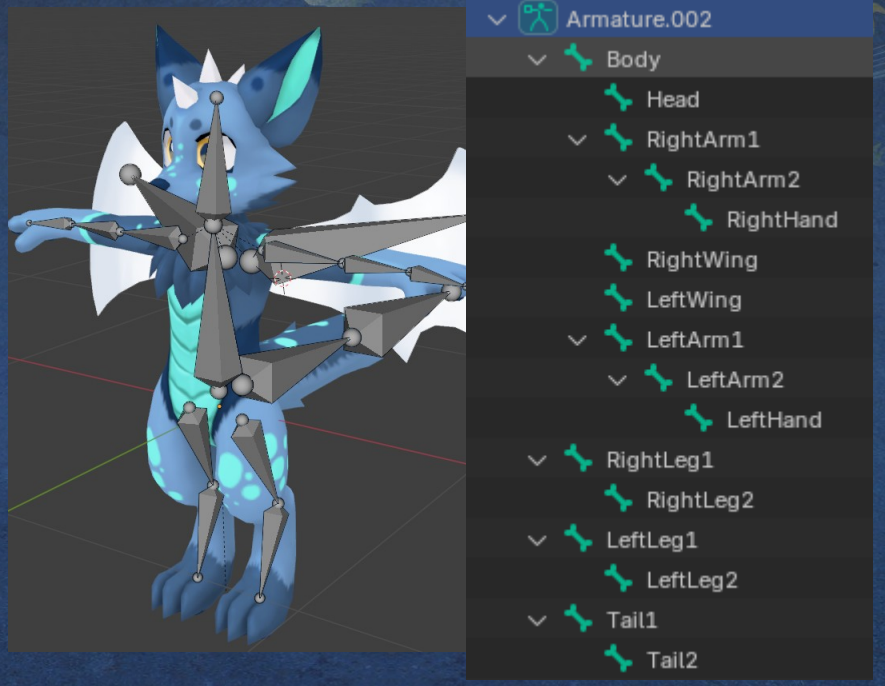
- Computer Graphics Programming in OpenGL with Java – second edition
  - Worked through the book some time in high school
- Moved to following an OpenGL in Java tutorial series by ThinMatrix
- Invaluable resource for learning computer graphics (not just in Java)
- Covers a huge range of topics and concepts, from basic rendering and the graphics pipeline to procedural generation, advanced lighting, particles, post-processing, animation, GUI, …





Skeletal Animation: Bones

- Ain't much of a skeleton without bones
- Will be using Bones and Joints interchangably
- Skeleton is just a hierarchy of bones, each with a translation, rotation and scale
- Each Vertex can be influenced by some number of bones by a varying amount
- Each bone is converted into a transformation matrix
- When rendered, the vertex position is multiplied by the transformation matrices of its associated bones, scaled by the weight, and then summed




```rust
#[repr(C)]
#[derive(Clone, Copy, Serialize, Deserialize, Debug)]
pub struct Vertex {
    pub pos: Vec3,
    pub norm: Vec3,
    pub tc: Vec2,
    /// Indices for which joints influence this vertex
    pub joints: [u8; 3],
    /// How much each of the 3 joints influence this vertex
    pub weights: Vec3,
}
```

In the Shader

- Need to load an array of transformation matrices for the bones
- Use an array of uniforms
- What is a uniform?
- Remember those shader assembly registers..?

c0-c95	vector	float	Read only	Application/Vertex-stream	Floating-point Constant registers.

- A “uniform” is a value that can be loaded into these constant registers before the shader is run, as a way to pass in data. We can load each joint’s transformation matrix into a set of these registers
- Each matrix will take 4 registers, as each register is a vector of 4 values, and we need 4 x 4 floats for the matrix
- Assuming some of these registers will be taken up by e.g. the model-view-projection matrix, the remaining ~80 registers gives us a hardware limit of around 20 bones/joint transformations
- But if they are loaded into registers instead of addressable memory, how do you index an array of matrices..?

Challenge: Indexing an Array

- In Picasso, there is syntactic sugar that makes it look like you are dealing with arrays, but each index is just an alias to one of the registers c0 – c95
- So you can't put a variable in the index spot..
- But you *kind of* can??
- The relative-addressing register a0 can store an integer and be used as an offset when reading from the constant registers
- So to index the joint transformations we must store the index in a0.x, and use that as an offset when reading the uniforms
- Note: Because this is assembly, a whole matrix multiplication is done with 4 dp4 (dot-product) instructions
- Now to actually load an animated model into our program

```
29  ; Uniforms
28  .fvec projection[4]
27  .fvec jointTransforms[64]
```

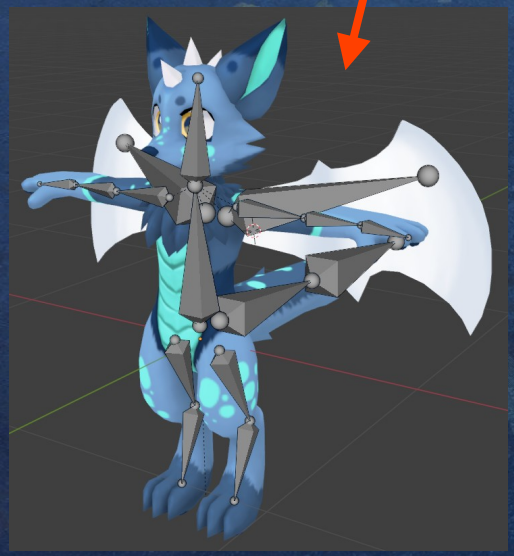
```
    ; outpos = projectionMatrix * localPos
    dp4 outpos.x, projection[0], r1
    dp4 outpos.y, projection[1], r1
    dp4 outpos.z, projection[2], r1
    dp4 outpos.w, projection[3], r1
```

```
; Animation transformations
; For each joint id
    ; Joint 1
    ; Address offset = joint_id * 4
    mul r3.x, nums.w, joint_ids.x
    mova a0.x, r3.x

    ; posePosition = transform[joint_id * 4 + n] * inpos: r3
    dp4 r3.x, jointTransforms[a0.x + 0], r0
    dp4 r3.y, jointTransforms[a0.x + 1], r0
    dp4 r3.z, jointTransforms[a0.x + 2], r0
    dp4 r3.w, jointTransforms[a0.x + 3], r0

    ; posePosition *= weight
    mul r3, joint_weights.xxxx, r3

    ; localPos += posePosition
    add r1, r1, r3
```

c0-c95	vector	float	Read only	Application/Vertex-stream	Floating-point Constant registers.
a0.x & a0.y	scalar	integer	Use/Write	Vertex shader	Address registers.

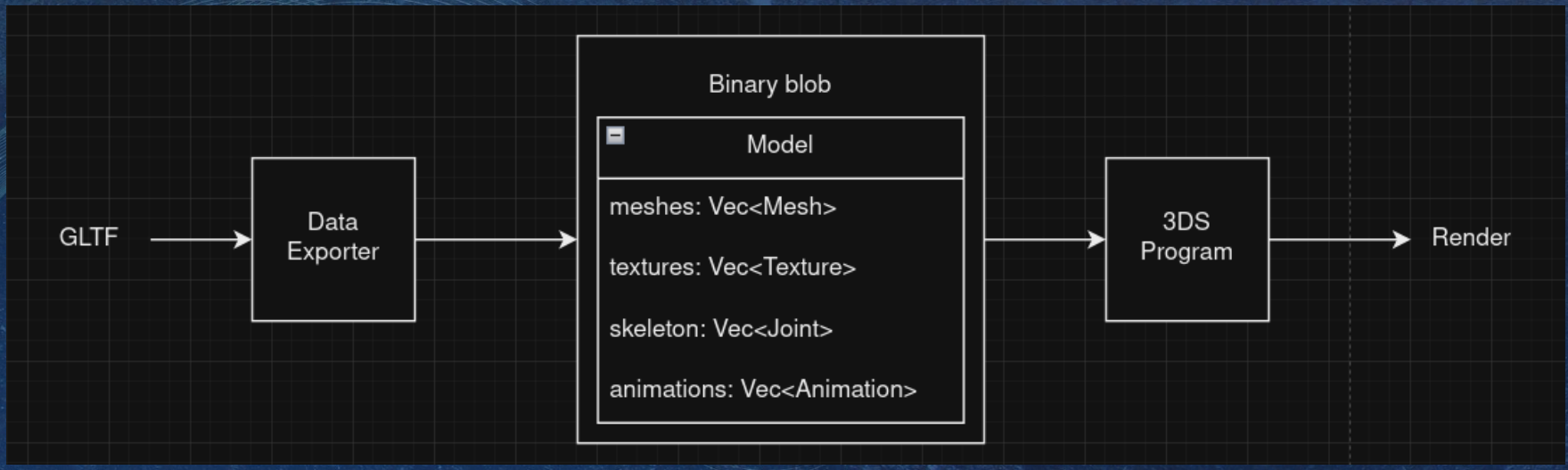
Challenge: Blender

- My model is wayyyy too complicated (84 bones, recall hardware limit of 20)
- Needed to learn to re-rig the model, brought it down to 16 bones
- Also decimated the mesh to only about 4k faces
- Learned to create a basic animation
- Needed a file format to export to that could contain all the model, skeleton, animation, and preferably texture information in one
- GLTF/GLB
- Becoming a standard portable format for 3D scenes

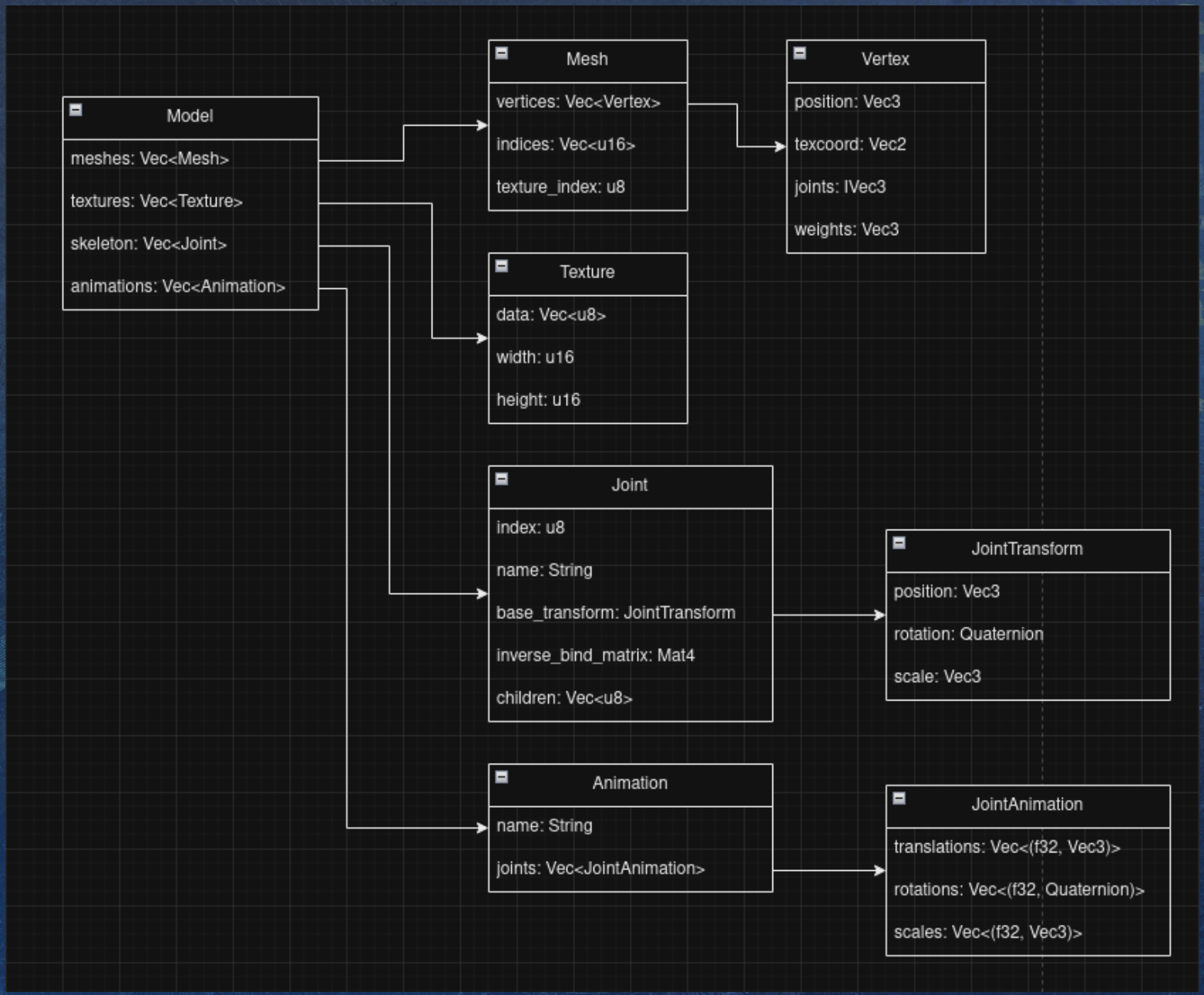


Blender to Render

- Could include the exported GLTF file in my 3DS program and parse everything at runtime? Sounds like a pain, would probably be slow and add a long latency when starting the program
- Instead: Introduce an intermediate library for all the data, containing everything I will need in the 3DS program and in the correct layout and format
- Include the library in a separate program which will parse the GLTF, format the data, and serialize the entire bundle into a binary representation of it
- Include the library in the 3DS program, and deserialise the binary into the ready-to-go data that can be sent directly to the GPU



Blender to Render: Data



Challenge: Texture Layout

- Textures are not linearly layed out like you expect a bitmap to be
- Follows a layout that is *similar* to a Z-Order curve
- But is not a standard z-order curve!
- Almost no documentation on what the actual order is??
- Blocks of about 8x8 are arranged in a hard-coded order
- Eventually found this snippet that conveniently calculates the correct index

Z-Order Curve

```
let dst_idx: usize = ((((y >> 3) * (data.width >> 3) + (x >> 3)) << 6)
    + ((x & 1)
        | ((y & 1) << 1)
        | ((x & 2) << 1)
        | ((y & 2) << 2)
        | ((x & 4) << 2)
        | ((y & 4) << 3))) as usize;
```

Has some wacky results if the texture isn't swizzled properly

Silly: Why is it wobbly??

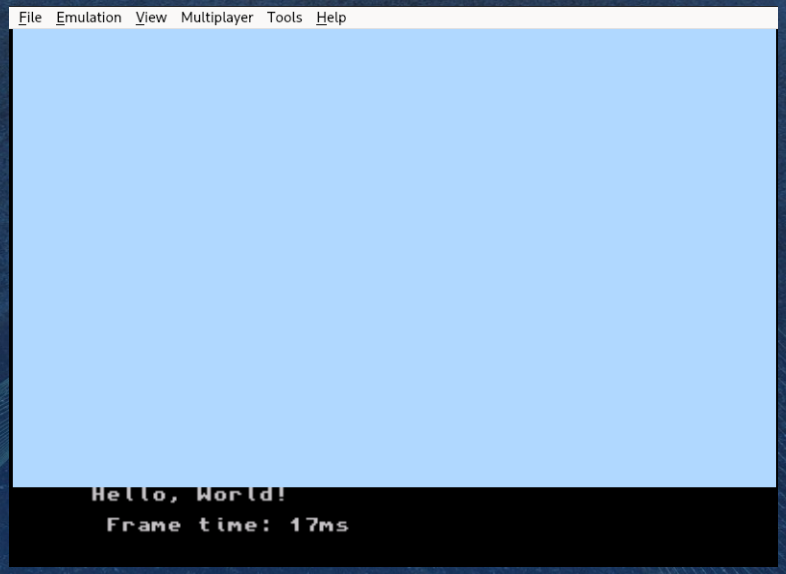
- Once I fixed the texture my bones seemed to be not quite right but close enough that something wasn't drastically wrong???

```
14  .proc main
13      ; Force the w component of inpos to be 1.0
12      ; r0 = inpos
11      mov r0.xyz, inpos
10      mov r0.w,   ones
 9
 8      ; localPos = Vec4(0.0, 0.0, 0.0, 1.0): r1
 7      mov r1.xyz, zeros
 6      mov r1.w,   ones
 5
```

It was because I set this w component to 1

Silly: Model disappeared once trying to use a pose from the animation??

- Applying the base pose worked, applying arbitrary transforms worked, as soon as the transforms were sampled from the animation nothing showed :(



Oops, all NaN!

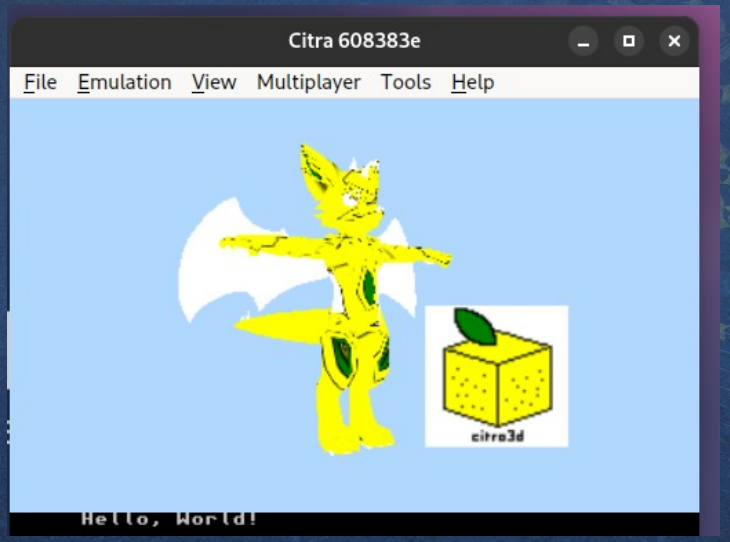
```
Extracting primitive
Extracting primitive
Extracting skeleton
Base pose:
[JointTransform { pos: Vec3(0.0, 0.06007162, 0.0)
t(-0.04697965, -3.9507333e-5, -0.0023827045, 0.99
274, -0.0009612278, -0.71743965, 0.69562685), sca
885, 0.0012741714, 0.020709815, 0.99950784), scal
93516, -0.005937278, -0.04510446, 0.99769783), sc
 -0.44055673, -0.3173654, 0.6461194), scale: Vec3
3173654, 0.6461194), scale: Vec3(1.0, 1.0, 1.0) }
98), scale: Vec3(0.99999994, 1.0, 1.0) }, JointTr
Vec3(1.0, 1.0, 1.0) }, JointTransform { pos: Vec3
1.0000001, 0.99999994) }, JointTransform { pos: V
1.0, 1.0000001) }, JointTransform { pos: Vec3(7.4
994, 0.9999999, 0.9999999) }, JointTransform { po
, 1.0000001, 1.0) }, JointTransform { pos: Vec3(1
.0000001, 1.0000002) }, JointTransform { pos: Vec
(0.99999994, 1.0, 1.0) }, JointTransform { pos: V
0, 1.0, 1.0) }]


Animated pose:
[JointTransform { pos: Vec3(NaN, NaN, NaN), rot:
aN, NaN), scale: Vec3(NaN, NaN, NaN) }, JointTran
3(NaN, NaN, NaN), rot: Quat(NaN, NaN, NaN, NaN),
 NaN, NaN) }, JointTransform { pos: Vec3(NaN, NaN
t(NaN, NaN, NaN, NaN), scale: Vec3(NaN, NaN, NaN)
rm { pos: Vec3(NaN, NaN, NaN), rot: Quat(NaN, NaN
le: Vec3(NaN, NaN, NaN) }, JointTransform { pos:
aN), rot: Quat(NaN, NaN, NaN, NaN), scale: Vec3(N
 JointTransform { pos: Vec3(NaN, NaN, NaN), rot:
aN, NaN), scale: Vec3(NaN, NaN, NaN) }, JointTran
Wrote data out to /home/eden/RustProjs/r3ds/asset
```

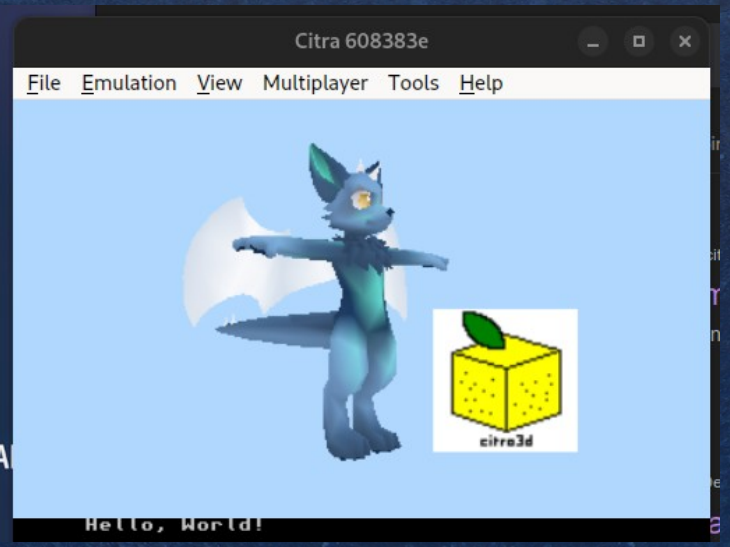
Silly: Bone hierarchy not in the correct order

- Bones had to be ordered such that when calculating the transformation matrices, the parents were visited before the child nodes
- At least for the iterative method I wanted to use, didn't want a recursive implementation
- Meant the bones were not in the same order their "indices" specified
- And so child nodes pointed to the wrong ones…
- Behold, cursed abominations --→
- (Had to remap the children field of each node to point to the correct indices)

Silly: Why is it using the wrong texture???

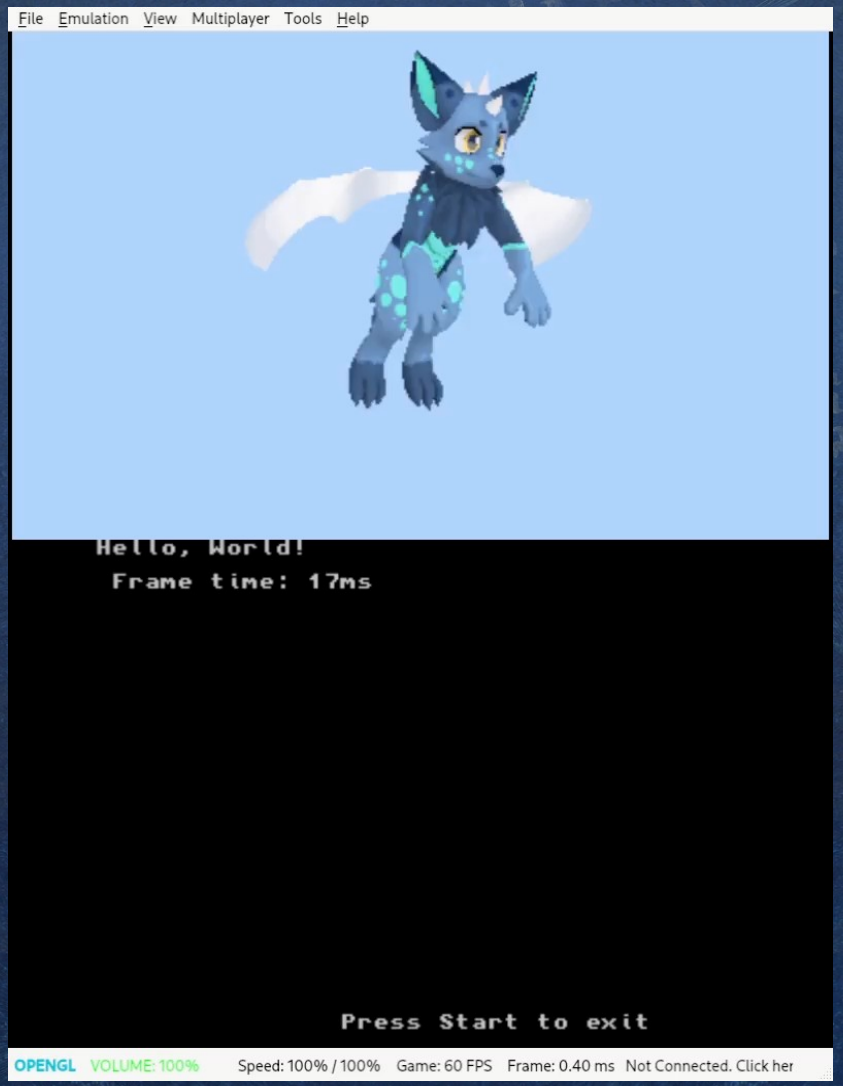
- Trying to use multiple textures for different objects in a single frame didn't seem to do anything..
- It always used the first texture despite changing them out in the TexEnv??
- Changing the texenv requires it be marked dirty...





But after all that...

- Create an animated model in Blender
- Export as GLTF/GLB
- Run it through the extractor
- Include the binary in the 3DS program



And it works!!