High-performance real-time 3D graphics with Vulkan (and Ruby)

Frederico de Oliveira Linhares

RubyKaigi, 2023
Ruby is simple in appearance, but is very complex inside, just like our human body.

Yukihiro Matsumoto
The Ruby way of doing 3D graphics

def init
    texture = CandyGear::Texture.from_image("texture.qoi")
    mesh = CandyGear::Mesh.new("meshe.cgmesh")
    $model = CandyGear::Model.new(mesh, texture)
    $instance = {
        position: CandyGear::Vector3D.new(0.0, 0.0, 0.0),
        rotation: CandyGear::Rotation3D.new(0.0, 0.0, 0.0)}
    view = CandyGear::View3D.new(
        CandyGear::Vector4D.new(0, 0, 1280, 720), 640, 360);
    view.camera_position =
        CandyGear::Vector3D.new(0.0, 0.0, 0.0)
end

def tick = $model.draw(  
    $instance[:position], $instance[:rotation])
Why Candy Gear is valuable for game developers

- Unreal and Unity are bloated; Candy Gear aims to be a thinner alternative.
- Ruby is multi-purpose, while Blueprint Visual Scripting and Godot Script are ad-hoc.
- It is hard to use Blueprint Visual Scripting with version control.
Why Candy Gear is suitable for the Ruby community

- Expand Ruby beyond the web industry.
- Test the Ruby language outside of web development.
- Attract game developers to the Ruby community, making the language more popular.
Engine implementation

**YARV vs mruby**

**YARV** the Ruby code calls the C/C++ code (the engine); therefore, every internal state in the framework is attached to Ruby objects.

**mruby** the C/C++ code (the engine) calls the Ruby code; therefore, it is easier to decouple the internal states of the framework from the Ruby environment.
Hollywood principle
a.k.a.: “Inversion of control”

Don’t call us, we’ll call you.

Hollywood principle
How Candy Gear executes a game

# Called once before the graphics engine is loaded
def config; end

# Called once after the engine is fully loaded
def init; end

# When you press a key
def key_down key; end

# When you release a key
def key_up key; end

# When you quit the game
def quit; end

# At the beginning of each frame
def tick; end
Multithreading and multiprocessing
at CPU level

- Starting and finishing threads are expensive operations.
- Create all threads at startup.
- Destroy all threads at finalization.
- Each thread gets and executes jobs from a job queue.
- If there are no jobs, the thread waits for new ones.
- There should not be more threads than processors.
Vulkan initialization

- Vulkan instances
- logical devices
- queues (for GPU multiprocessing)
- swap chain
- graphics pipelines
Command buffers

- Queues execute work.
- Work must be inside command buffers.
- Candy Gears caches drawing commands.
- Queues work in parallel.
Commonly, a 3D engine has pipelines for:

- 3D models (can be split into several pipelines);
- 2D sprites;
- sky;
- (sea, pool, or river) water.
3D graphics pipeline
(for 3D models)

Vertex

- **textured model** coordinates x, y, z, u, v
- **wireframe model** coordinates x, y, z

Polygon

A polygon consists of a sequence of 3 vertexes.

Mesh

A mesh consists of a sequence of one or more vertexes.

Vertex index

Create an index for each vertex to reuse the vertexes.
**Sprite**

- A sprite is a rectangular region from a texture.
- A graphic card renders a sprite as 2 polygons.
- It is cheaper to render 2D polygons.
Sky graphics pipeline
(cube maps and skyboxes)

Cube map

- It uses six different textures as one cube.
- It samples all images sampled as one.
- It uses a 3D vector to calculate which cube regions to display.
Loading data into a graphics card

```ruby
image = CandyGear::Texture.from_image("image.qoi")
mesh = CandyGear::Mesh.new("object.cgmesh")
font = CandyGear::Font.new("font.ttf", 16)
```

- Load data from disk
- Decompress data
- Convert data into the format required by the graphics card
- Move the data to the graphics card
- Return an Ruby object that handles the data into the graphics card
Descriptor sets

World:
- ambient light (vertex shader);
- and directional light (fragment shader).

View:
- camera position;
- camera rotation;
- and projection.

Model Instance:
- instance position;
- and instance rotation;

\[
\text{model.draw(}
\quad \text{instance_position, instance_rotation)}
\]
## Graphics pipeline stages

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Draw</strong></td>
<td>This stage receives the commands for the graphics pipeline.</td>
</tr>
<tr>
<td><strong>Input Assembly</strong></td>
<td>Assemble vertices from Vertex Buffer and (optionally) Index Buffer into geometric primitives based on topology.</td>
</tr>
<tr>
<td><strong>Vertex Shader</strong></td>
<td>Converts vertex position to screen position.</td>
</tr>
</tbody>
</table>
### Tessellation (three stages)

Subdivide polygons into smaller polygons and apply transformations to the new generated polygons.

![Tessellation Diagram]

### Geometry Shader

Can subdivide, reduce and modify polygons. Similar to tessellation but with a different finality.

![Geometry Shader Diagram]

### Vertex Post-Processing

Assembly primitives produced by the previous stage (vertex shader, tessellation, or geometry shader). Clip and cull; discard parts of primitives that the rasterization will not use.

![Vertex Post-Processing Diagram]
Graphics pipeline stages

**Rasterizer**

Converts 3D coordinates into rasterized fragments.

**Fragment shader (three stages)**

Adds colors and depth to the fragments produced by the Rasterizer.

**Color Blending**

Mix all the fragments according to their transparency level and depth into one image, generating the frame displayed on the screen.
To-do

Graphics:
1. Finish multithread system (CPU).
2. Finish the graphical engine.
3. Create default file formats.

Audio:
- Create an audio engine.
- Create a MIDI synthesizer.

Ports:
1. Windows (with Vulkan)
2. Nintendo
3. PlayStation

Cleanup:
- Remove libSDL (it adds unnecessary indirections to the engine).
### References

| **Vulkan Tutorial** | It is a tutorial for absolute beginners in 3D graphics.  
https://vulkan-tutorial.com/ |
|---------------------|---------------------------------------------------------------------|
| **Vulkan Programming Guide** | This book presumes that the reader has some basic knowledge of 3D graphics. If you are entirely new, read the tutorial first.  
https://www.vulkanprogrammingguide.com/ |
| **Kohi Game Engine** | It is a series of videos teaching how to make a game engine from scratch in C.  
https://www.youtube.com/watch?v=dHPuU-DJoBM&list=PLv8Ddw9K0J Pg1BE0-RS-0MYs423cvLVtj |
| **Candy Gear Game Engine** | It is my game engine. It is written in C++ and uses Vulkan and mruby.  
https://bitbucket.org/fredlinhares/candygear |