Building WebGPU with Rust

Fosdem, 2th Feb 2020

Dzmitry Malyshau @kvark
(Mozilla / Graphics Engineer)
Agenda

1. WebGPU: Why and What?
2. Example in Rust
3. Architecture
4. Rust features used
5. Wrap-up
6. (bonus level) Browsers
Can we make this simpler?
Developers want to have rich content running portably on the Web and Native.

Each native platform has a preferred API.

Some of them are best fit for engines, not applications.

The only path to reach most platforms is OpenGL/WebGL.

- Applications quickly become CPU-limited.
- No multi-threading is possible.
- Getting access to modern GPU features portably is hard, e.g. compute shaders are not always supported.
OpenGL

Render like it's 1992

EVERY DAY

WE STRAY FURTHER FROM GPU
Future of OpenGL?

- Apple -> deprecates OpenGL in 2018, there is no WebGL 2.0 support yet
- Microsoft -> not supporting OpenGL (or Vulkan) in UWP
- IHVs focus on Vulkan and DX12 drivers
- WebGL ends up translating to Dx11 (via Angle) on Windows by major browsers
OptionGL: technical issues

- Changing a state can cause the driver to recompile the shader, internally
  - Causes 100ms freezes during the experience...
  - Missing concept of pipelines
- Challenging to optimize for mobile
  - Rendering tile management is critical for power-efficiency but handled implicitly
  - Missing concept of render passes
- Challenging to take advantage of more threads
  - Purely single-threaded, becomes a CPU bottleneck
  - Missing concept of command buffers
- Tricky data transfers
  - Dx11 doesn’t have buffer to texture copies
- Given that WebGL2 is not universally supported, even basic things like sampler objects are not fully available to developers
OpenGL: evolution

GPU all the things!
Who started WebGPU?

Quiz^^

hint: not Apple
Need a better API on the Web

WebVulkan
WebMetal
Design a new idiomatic API for the Web?

3D Portability /WebGL-Next
Khronos Vancouver F2F
2019 Sep: Gecko implementation start

2018 Sep: wgpu project kick-off

2018 Apr: agreement on the implicit barriers

2017 Jun: agreement on the binding model

2017 Feb: formation of W3C group

2016 H2: experiments by browser vendors
What is WebGPU?
How standards proliferate

(insert XKCD #927 here)

WebGPU on native?
Design Constraints

security   portability   performance   usability
<table>
<thead>
<tr>
<th>Fish Number</th>
<th>Dawn/D3D12</th>
<th>Dawn/Vulkan</th>
<th>Native D3D12</th>
<th>Optimized OpenGL FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fish Count</td>
<td>Non-MSAA/MSAA</td>
<td>Non-MSAA/MSAA</td>
<td>Non-MSAA/MSAA</td>
<td>Non-MSAA/MSAA</td>
</tr>
<tr>
<td>50000</td>
<td>27/22</td>
<td>43/34</td>
<td>40/32</td>
<td>33/26</td>
</tr>
<tr>
<td>30000</td>
<td>43/34</td>
<td>60/47</td>
<td>60/45</td>
<td>49/36</td>
</tr>
<tr>
<td>20000</td>
<td>60/47</td>
<td>60/57</td>
<td>60/57</td>
<td>64/44</td>
</tr>
</tbody>
</table>

Windows 10 17134, Coffee lake, UHD 630

<table>
<thead>
<tr>
<th>Fish Number</th>
<th>Dawn/Metal</th>
<th>Optimized OpenGL FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fish Count</td>
<td>Non-MSAA/MSAA</td>
<td>Non-MSAA/MSAA</td>
</tr>
<tr>
<td>50000</td>
<td>33/31</td>
<td>26/26</td>
</tr>
<tr>
<td>30000</td>
<td>48/45</td>
<td>40/39</td>
</tr>
<tr>
<td>20000</td>
<td>64/62</td>
<td>62/56</td>
</tr>
</tbody>
</table>

MacOS 10.14.4, Intel core i5, AMD Radeon Pro 555X

<table>
<thead>
<tr>
<th>Fish Number</th>
<th>Dawn/Vulkan</th>
<th>Optimized OpenGL FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fish Count</td>
<td>Non-MSAA/MSAA</td>
<td>Non-MSAA/MSAA</td>
</tr>
<tr>
<td>50000</td>
<td>32/29</td>
<td>11/11</td>
</tr>
<tr>
<td>30000</td>
<td>55/53</td>
<td>19/17</td>
</tr>
<tr>
<td>20000</td>
<td>73/70</td>
<td>27/25</td>
</tr>
</tbody>
</table>

Ubuntu 19.04, Coffee lake, UHD 630

Early (native) benchmarks by Google
Early (web) benchmarks by Safari team

**Triangles Benchmark (Higher is Better)**

macOS Catalina Beta 7, STP 91

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Model</th>
<th>WebGL Speed</th>
<th>WebGPU Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>15” MacBook Pro (2016)</td>
<td>Integrated</td>
<td>1x</td>
<td>2.19x</td>
</tr>
<tr>
<td>15” MacBook Pro (2016)</td>
<td>Discrete</td>
<td>1x</td>
<td>4.49x</td>
</tr>
<tr>
<td>iMac Pro</td>
<td></td>
<td>1x</td>
<td>7.64x</td>
</tr>
</tbody>
</table>
Example: device initialization

```rust
let adapter = wgpu::Adapter::request(
    &wgpu::RequestAdapterOptions { power_preference: wgpu::PowerPreference::Default },
    wgpu::BackendBit::PRIMARY,
).unwrap();

let (device, queue) = adapter.request_device(&wgpu::DeviceDescriptor {
    extensions: wgpu::Extensions { anisotropic_filtering: false },
    limits: wgpu::Limits::default(),
});
```
Example: swap chain initialization

```rust
let surface = wgpu::Surface::create(&window);

let swap_chain_desc = wgpu::SwapChainDescriptor {
    usage: wgpu::TextureUsage::OUTPUT_ATTACHMENT,
    format: wgpu::TextureFormat::Bgra8UnormSrgb,
    width: size.width,
    height: size.height,
    present_mode: wgpu::PresentMode::Vsync,
};

let mut swap_chain = device.create_swap_chain(&surface, &swap_chain_desc);
```
Example: uploading vertex data

```rust
let vertex_buf = device.create_buffer_with_data(vertex_data.as_bytes(), wgpu::BufferUsage::VERTEX);

let vb_desc = wgpu::VertexBufferDescriptor {
    stride: vertex_size as wgpu::BufferAddress,
    step_mode: wgpu::InputStepMode::Vertex,
    attributes: &[wgpu::VertexAttributeDescriptor { format: wgpu::VertexFormat::Float4, offset: 0, shader_location: 0 }, wgpu::VertexAttributeDescriptor { format: wgpu::VertexFormat::Float2, offset: 4 * 4, shader_location: 1 }],
};
```
Is WebGPU an explicit API?

Quiz ^

hint: what is explicit?
Feat: implicit memory

WebGPU:
```javascript
texture = device.createTexture({..});
```

Vulkan:
```c
image = vkCreateImage();
reqs = vkGetImageMemoryRequirements();
memType = findMemoryType();
memory = vkAllocateMemory(memType);
vkBindImageMemory(image, memory);
```
Example: declaring shader data

```rust
let bind_group_layout = device.create_bind_group_layout(&wgpu::BindGroupLayoutDescriptor {
    bindings: &[
        wgpu::BindGroupLayoutBinding {
            binding: 0,
            visibility: wgpu::ShaderStage::VERTEX,
            ty: wgpu::BindingType::UniformBuffer { dynamic: false },
        },
    ],
});

let pipeline_layout = device.create_pipeline_layout(&wgpu::PipelineLayoutDescriptor {
    bind_group_layouts: &[&bind_group_layout],
});
```
Example: instantiating shader data

```rust
let bind_group = device.create_bind_group(&wgpu::BindGroupDescriptor {
    layout: &bind_group_layout,
    bindings: &[wgpu::Binding {
        binding: 0,
        resource: wgpu::BindingResource::Buffer {
            buffer: &uniform_buf,
            range: 0..64,
        },
    },],
});
```
Feat: binding groups of resources

- Vertex buffer 0 to Vertex buffer 1
- Shaders
  - Bind Group 0
  - Uniform buffer
  - Sampled texture
  - Bind Group 1
  - Bind Group 2
  - Bind Group 3
  - Storage buffer
  - Sampler
- Render Target 0
- Render Target 1
Example: creating the pipeline

```rust
let pipeline = device.create_render_pipeline(&wgpu::RenderPipelineDescriptor {
    layout: &pipeline_layout,
    vertex_stage: wgpu::ProgrammableStageDescriptor { module: &vs_module, entry_point: "main" },
    fragment_stage: Some(wgpu::ProgrammableStageDescriptor { module: &fs_module, entry_point: "main" }),
    rasterization_state: Some(wgpu::RasterizationStateDescriptor { front_face: wgpu::FrontFace::Ccw, cull_mode: wgpu::CullMode::Back }),
    primitive_topology: wgpu::PrimitiveTopology::TriangleList,
    color_states: &[wgpu::ColorStateDescriptor { format: sc_desc.format, ... }],
    index_format: wgpu::IndexFormat::Uint16,
    vertex_buffers: &[wgpu::VertexBufferDescriptor {
        stride: vertex_size as wgpu::BufferAddress,
        step_mode: wgpu::InputStepMode::Vertex,
        attributes: &[
            wgpu::VertexAttributeDescriptor { format: wgpu::VertexFormat::Float4, offset: 0, shader_location: 0 },
        ],
    }],
});
```
Example: rendering

```rust
let mut rpass = encoder.begin_render_pass(&wgpu::RenderPassDescriptor {
    color_attachments: &[wgpu::RenderPassColorAttachmentDescriptor {
        attachment: &frame.view,
        resolve_target: None,
        load_op: wgpu::LoadOp::Clear,
        store_op: wgpu::StoreOp::Store,
        clear_color: wgpu::Color { r: 0.1, g: 0.2, b: 0.3, a: 1.0 },
    }],
    depth_stencil_attachment: None,
});

rpass.set_pipeline(&self.pipeline);

rpass.set_bind_group(0, &self.bind_group, &[]);

rpass.set_index_buffer(&self.index_buf, 0);

rpass.set_vertex_buffers(0, &[(&self.vertex_buf, 0)]);

rpass.draw_indexed(0 .. self.index_count as u32, 0, 0 .. 1);
```
Feat: render passes
Feat: multi-threading

Command Buffer 1 (recorded on **thread A**)
- Render pass
  - setBindGroup
  - setVertexBuffer
  - draw
  - setIndexBuffer
  - drawIndexed

Command Buffer 2 (recorded on **thread B**)
- Compute pass
  - setBindGroup
  - dispatch

Submission (on **thread C**)
- Command buffer 1
- Command buffer 2
Example: work submission

```rust
let mut encoder = device.create_command_encoder(
    &wgpu::CommandEncoderDescriptor::default()
);

// record some passes here
let command_buffer = encoder.finish();

queue.submit(&[command_buffer]);
```
**Feat: implicit barriers**

Tracking resource usage

**Command stream:**
- RenderPass-A {..}
- Copy()
- RenderPass-B {..}
- ComputePass-C {..}

**Texture usage**
- OUTPUT_ATTACHMENT
  - COPY_SRC
  - SAMPLED
  - STORAGE

**Buffer usage**
- STORAGE_READ
  - COPY_DST
  - VERTEX + UNIFORM
  - STORAGE

Space for optimization
Is WSL the chosen shading language?

hint: what is WSL?
API: missing pieces

- Shading language
- Multi-queue
- Better data transfers
Is WebGPU only for the Web?

Quiz:

hint: what is explicit?
Demo time!
Graphics Abstraction

- Vulkan
- Vulkan PORTABLE
- gfx.rs
- DirectX 12
- DirectX 11
- OpenGL
- OpenGL ES
Problem: contagious generics

```rust
struct Game<B: hal::Backend> {
    sound: Sound,
    physics: Physics,
    renderer: Renderer<B>,
}
```
Solution: backend polymorphism

Impl Context {

    pub fn device_create_buffer<B: GfxBackend>(&self, ...) { … }

}

#[no_mangle]

pub extern "C" fn wgpu_server_device_create_buffer(

    global: &Global, self_id: id::DeviceId, desc: &core::resource::BufferDescriptor, new_id: id::BufferId

) {

    gfx_select!(self_id => global.device_create_buffer(self_id, desc, new_id));

}
Identifiers and object storage

Index (32 bits)  Epoch (29 bits)  Backend (3 bits)


Vulkan backend
Usage tracker

**Tracker**

- Index (32 bits)
- Epoch
- Ref Count
- State

**State**

- Subresource
- Usage
Usage tracking: sync scopes

Old -> Expected -> New
Usage tracking: merging

Union

Bind Group

Render Pass

Command Buffer

Device

Replace

Compute
Usage tracking: sub-resources
Usage tracking: simple solution

```rust
pub struct Unit<U> {
    first: Option<U>,
    last: U,
}
```
Lifetime tracking

Command buffer tracker

Bind group

Resource

User

Device

GPU in flight

Submission 1

Submission 2

Submission 3

Last used
Wgpu-rs: project structure

- wgpu-rs
- wgpu-native
- wgpu-core
- web-sys
- Emscripten
- Dawn
pub enum BindingType {
    UniformBuffer { dynamic: bool },
    StorageBuffer { dynamic: bool, readonly: bool },
    Sampler,
    SampledTexture {
        multisampled: bool,
        dimension: TextureViewDimension,
    },
    StorageTexture { dimension: TextureViewDimension },
}
impl<'a> RenderPass<'a> {
    pub fn set_index_buffer(
        &mut self,
        buffer: &'a Buffer,
        offset: BufferAddress
    ) {...}
}
Wgpu-rs: Exclusive Encoding

```rust
pub struct CommandEncoder {
    id: wgc::id::CommandEncoderId,
    _p: std::marker::PhantomData</*const u8>>,
}

pub struct ComputePass<'a> {
    id: wgc::id::ComputePassId,
    _parent: &'a mut CommandEncoder,
}
```
Wgpu-rs: Borrowing

- Borrow all the things! (resource, swapchain, etc)
- C bindings with &borrowing
let mut token = Token::root();

let (device_guard, mut token) = 
    hub.devices.read(&mut token);

hub.pipeline_layouts
    .register_identity(id_in, layout, &mut token)
Wgpu: Ecosystem

- Parking_lot
- Gfx/Rendy
- VecMap/SmallVec/ArrayVec
- Cbindgen (for ffi)
- Winit (for examples)
- etc
The Bad

- Passing slices in the C API
- Generics aren’t always good (compile time, contagious, usability)
Future work

Web Target
Error handling
OpenGL backend
Optimization
Links

- [https://github.com/gpuweb/gpuweb](https://github.com/gpuweb/gpuweb) - upstream spec
- [https://github.com/gfx-rs/wgpu](https://github.com/gfx-rs/wgpu) - our implementation in Rust
- [https://github.com/gfx-rs/wgpu-rs](https://github.com/gfx-rs/wgpu-rs) - Rust API wrapper and examples
- [https://dawn.googlesource.com/dawn](https://dawn.googlesource.com/dawn) - Google’s implementation
- [https://github.com/webgpu-native/webgpu-headers](https://github.com/webgpu-native/webgpu-headers) - shared headers
Bonus: Browser Architecture
Overview

Content process

WebGPU bindings

wgpu-remote

GPU process

wgpu-core

gpu

gfx.rs
Buffer creation
javascript

setBindGroup  setVertexBuffer  draw

Content process

Poke
Pass command list
Resource dependencies

error

GPU process

Poke

vkCmdBindDescriptorSets
vkCmdBindVertexBuffer
vkCmdDraw

Pass recording
Are we WebGPU yet?

- Chromium: High completeness
- Safari: Moderate completeness
- Firefox: Low completeness
- Servo: Lowest completeness
Thank You!

- *gfx-rs/wgpu community* (contributions)
- *Joshua Groves* (reviews)
- *Mozilla gfx team*
- *Corentin Wallez* (feedback)