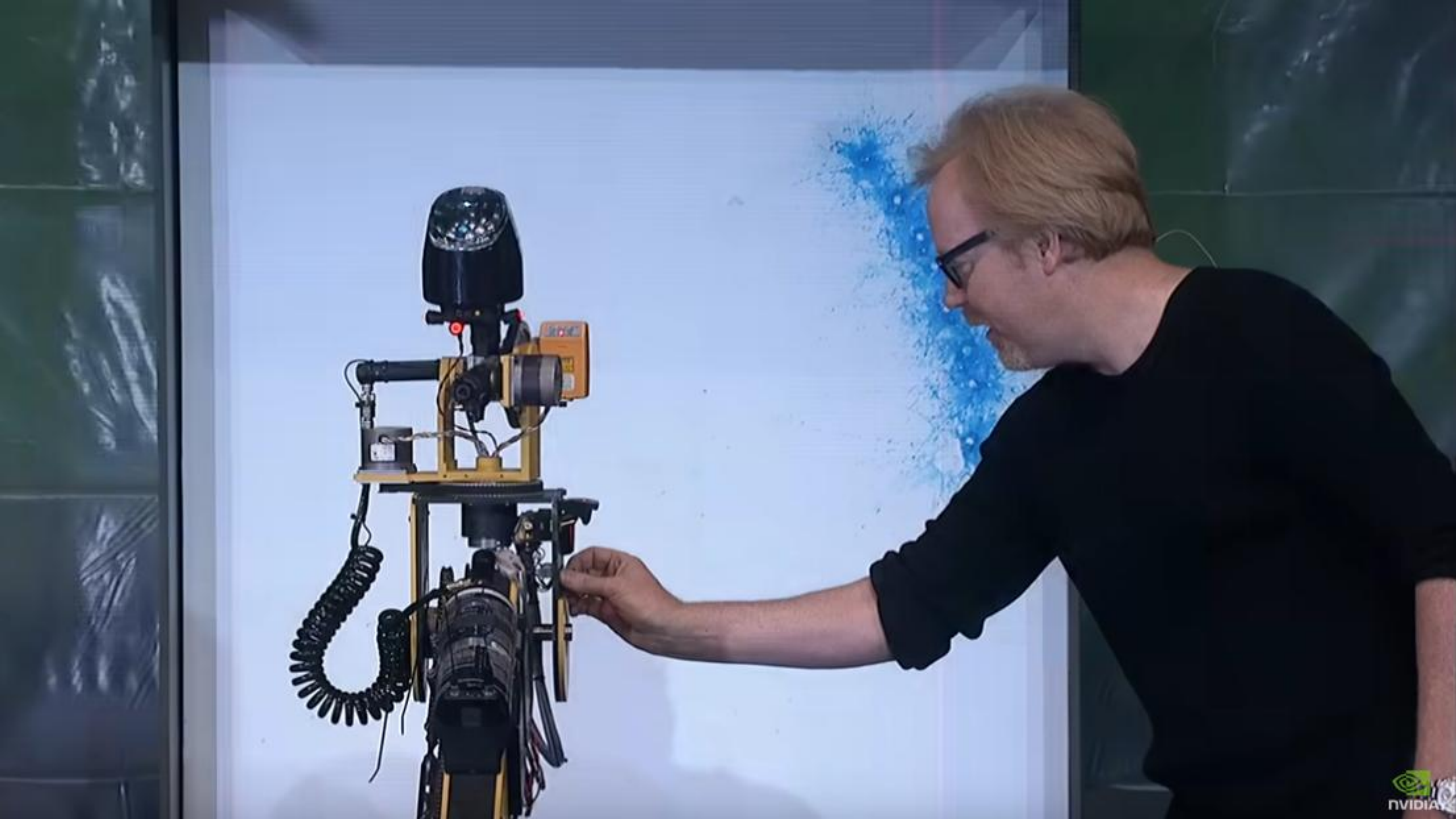
Rust on the GPU

Joey Nicholas

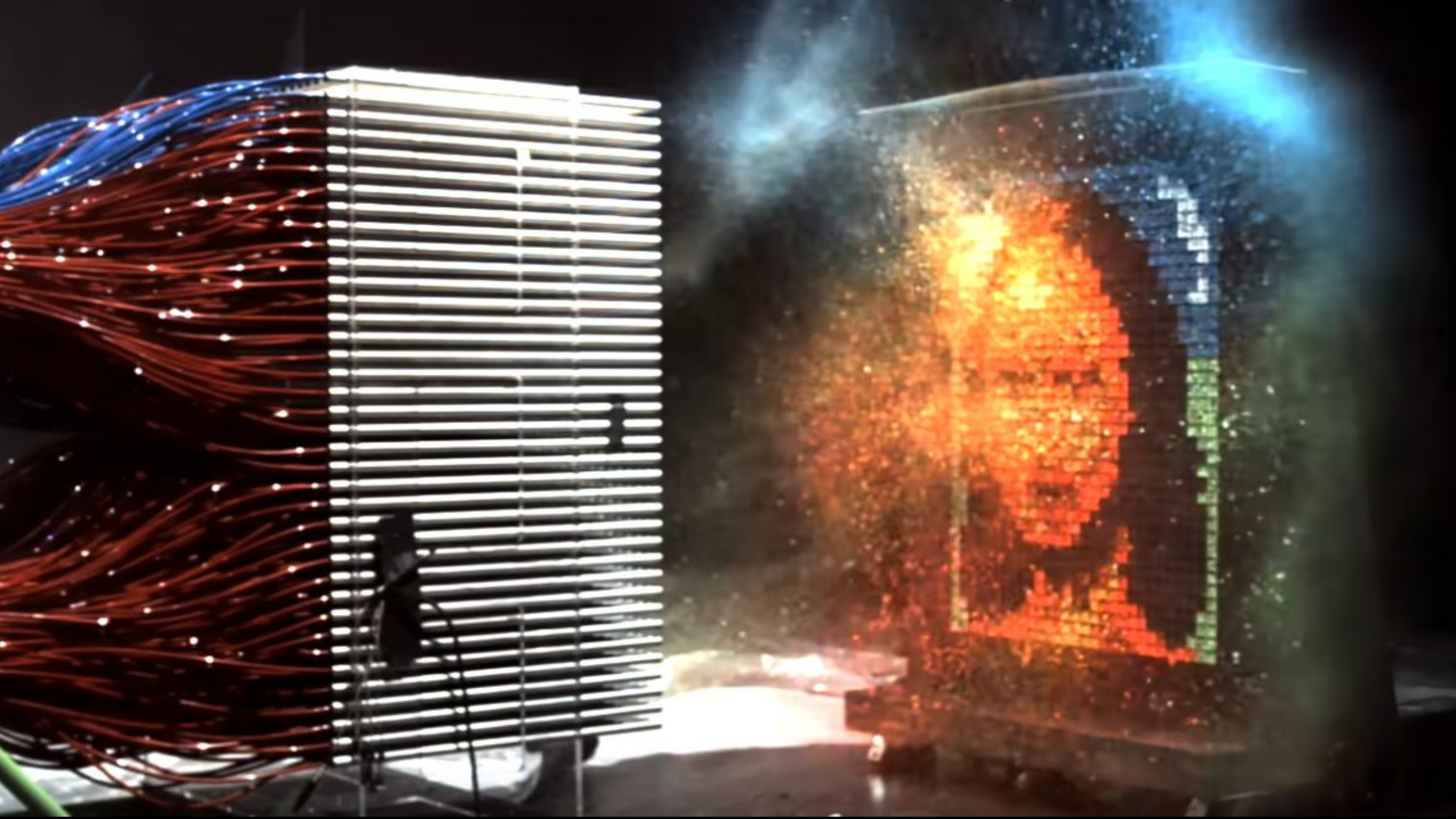
https://joeyn.dev

GPU

What is it?



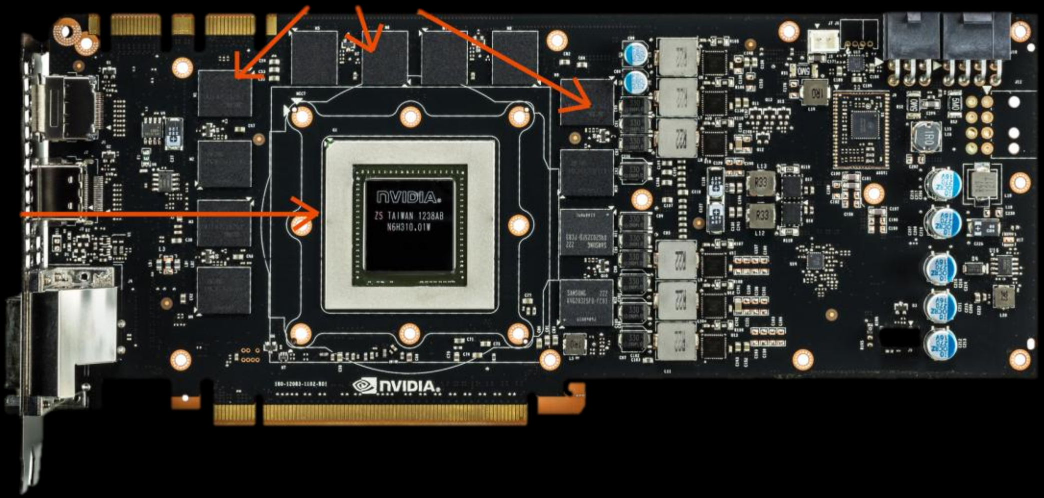
NVIDIA



CPU

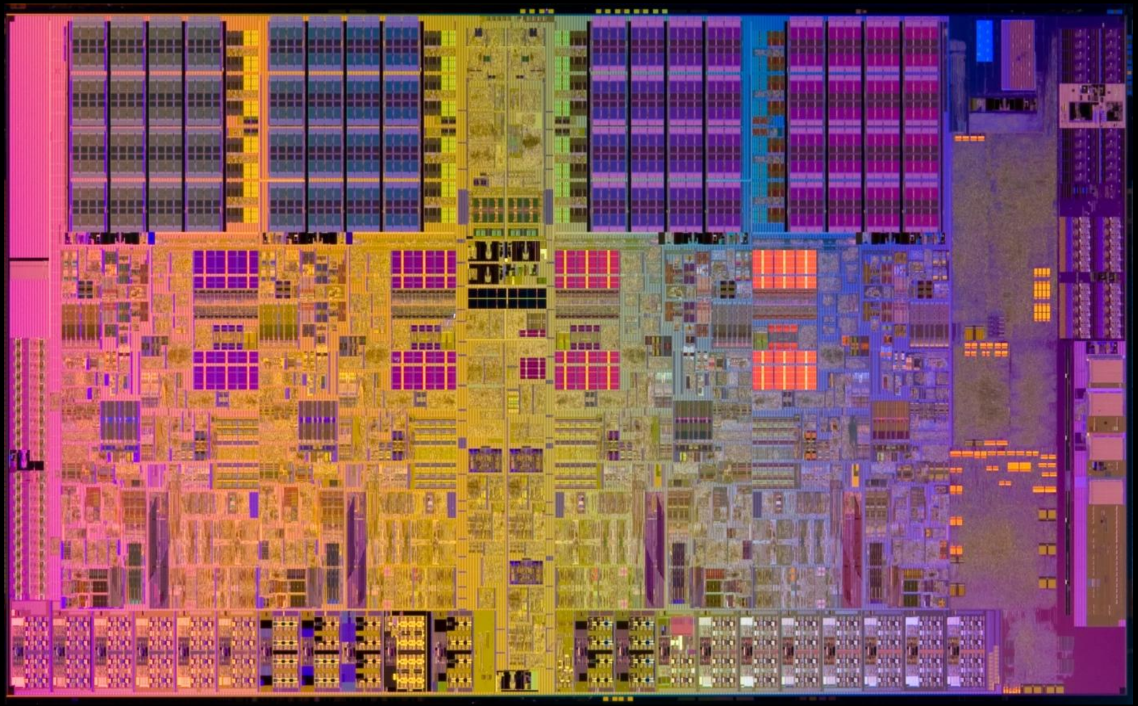
- A CPU, Wikipedia

GPU



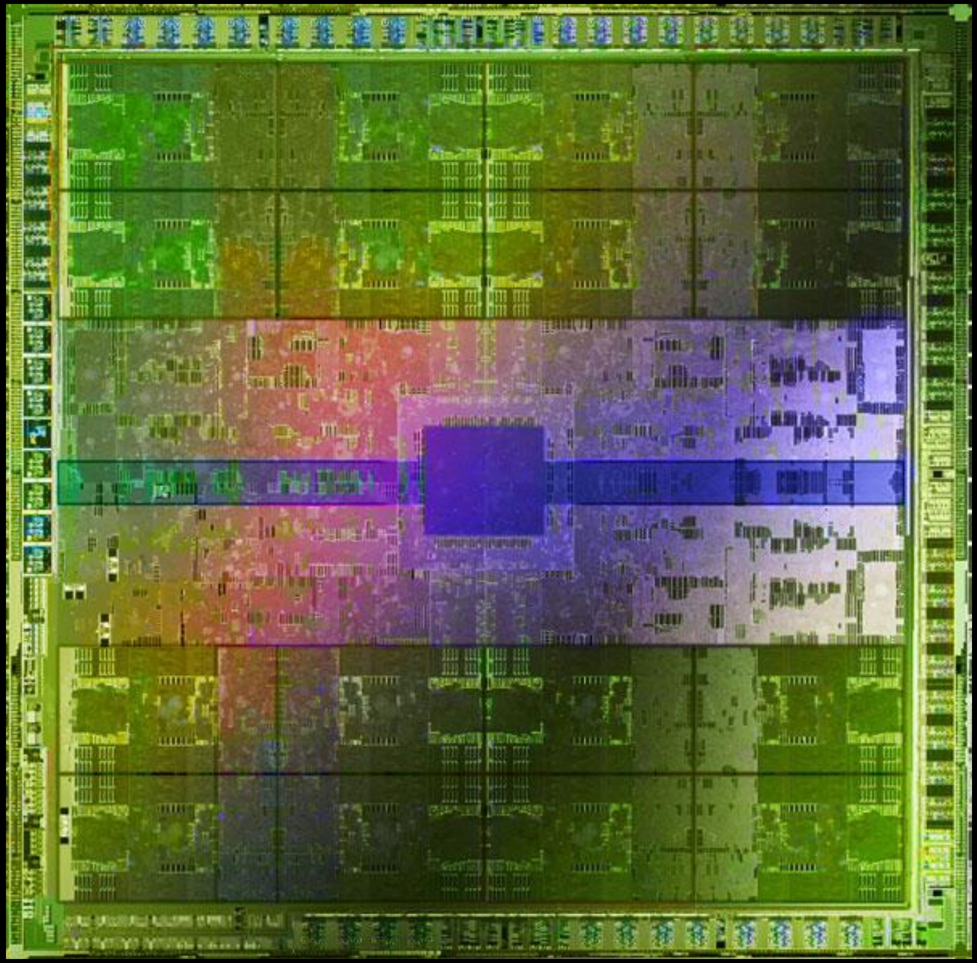
- GPU Memory bandwidth, DigitalOcean

CPU

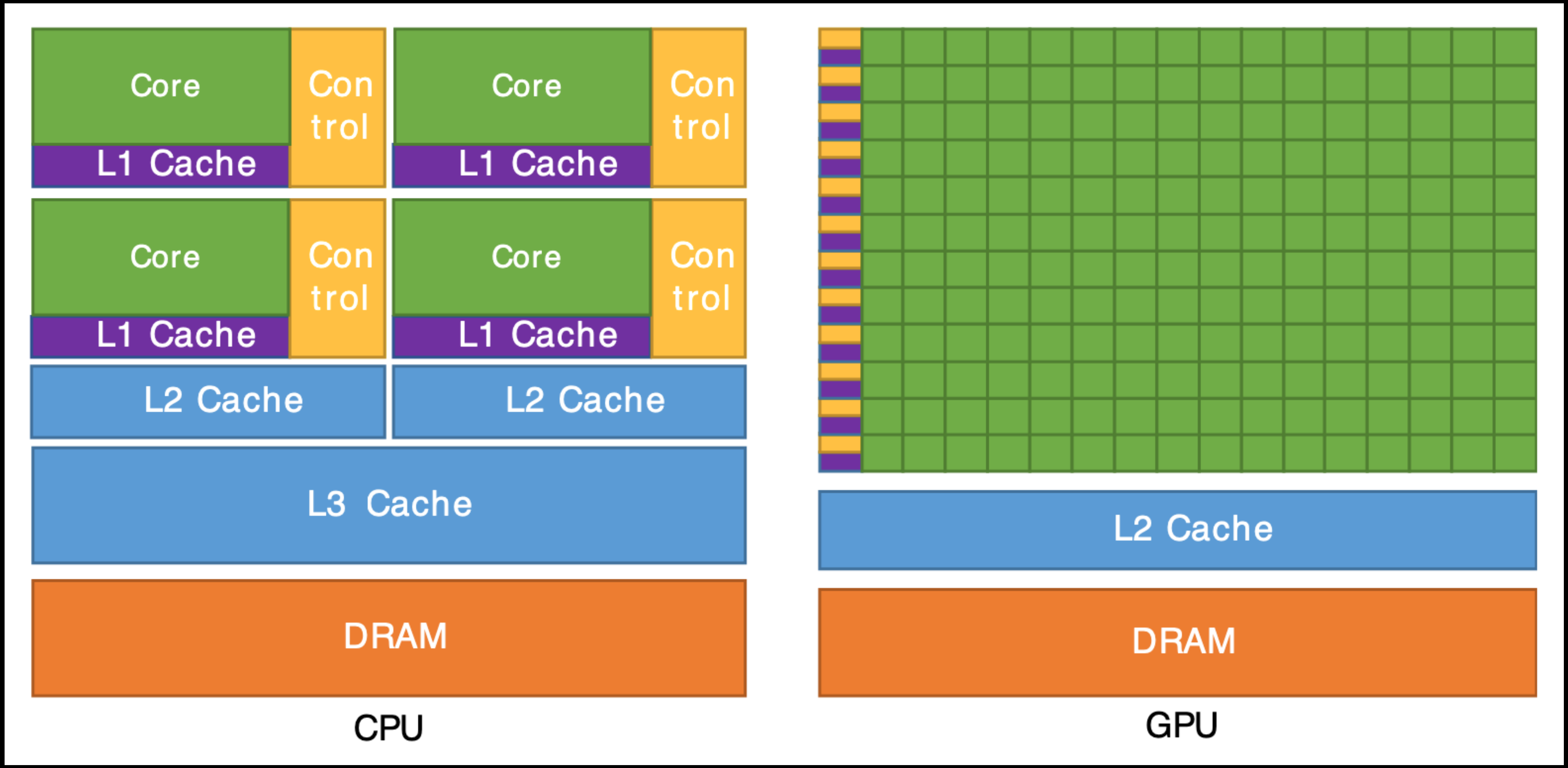


- CPU DIE SHOTS, The Higher Inquiètude

GPU



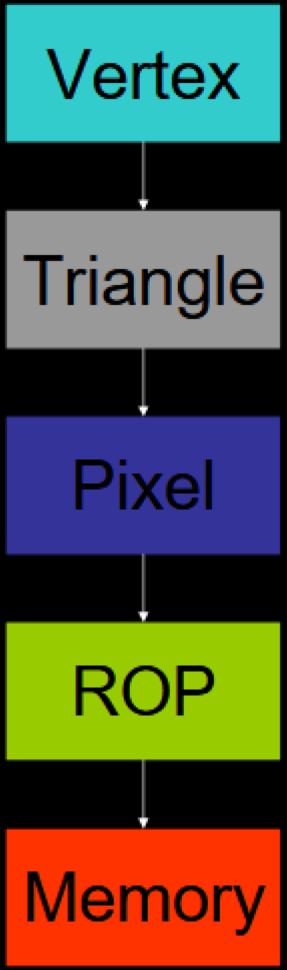
- GPU Dictionary: Understanding GPU & Video Card Specs, GamersNexus



- Design: GPU vs. CPU, Cornell Virtual Workshop

Graphics Pipelines For Last ~~20 Years~~ Pre 2006 -ish

Processor per function



Vertex

T&L evolved to vertex shading

Triangle

Triangle, point, line – setup

Pixel

Flat shading, texturing, eventually pixel shading

ROP

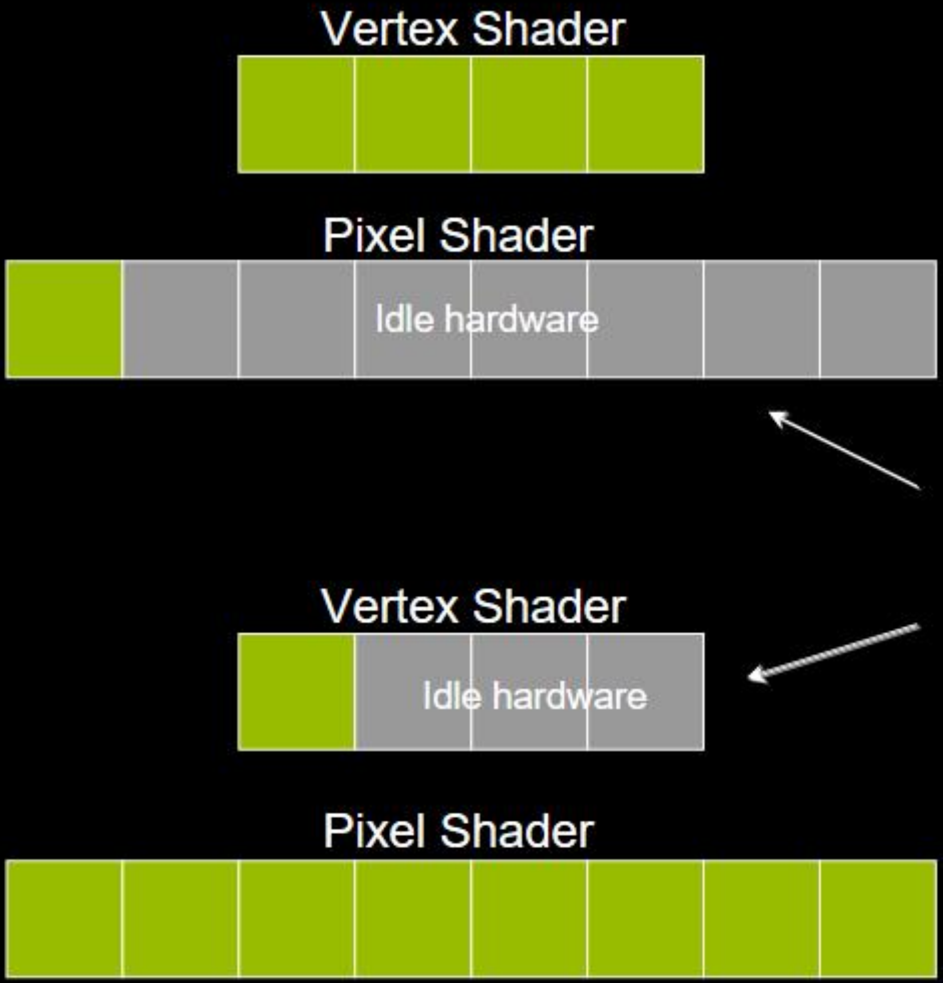
Blending, Z-buffering, antialiasing

Memory

Wider and faster over the years

- *An introduction to Modern GPU Architecture,* Ashu Rege

Why Unify?



Vertex Shader

Pixel Shader

Idle hardware

Vertex Shader

Idle hardware

Pixel Shader

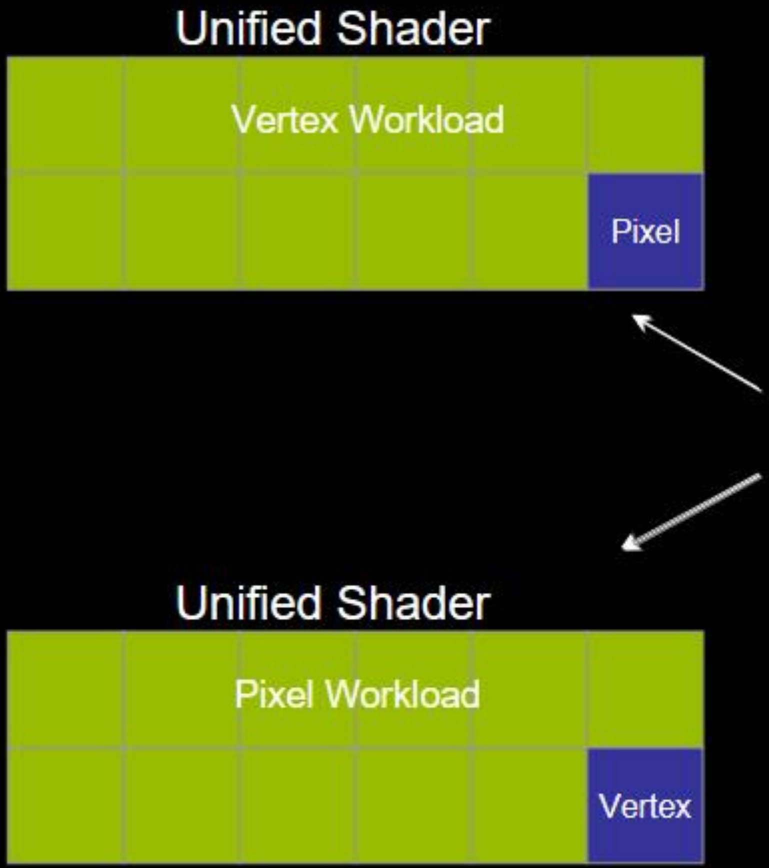
Unbalanced and inefficient utilization in non-unified architecture

Heavy Geometry
Workload Perf = 4

Heavy Pixel
Workload Perf = 8

- *An introduction to Modern GPU Architecture,* Ashu Rege

Why Unify?



Unified Shader

Vertex Workload

Pixel

Unified Shader

Pixel Workload

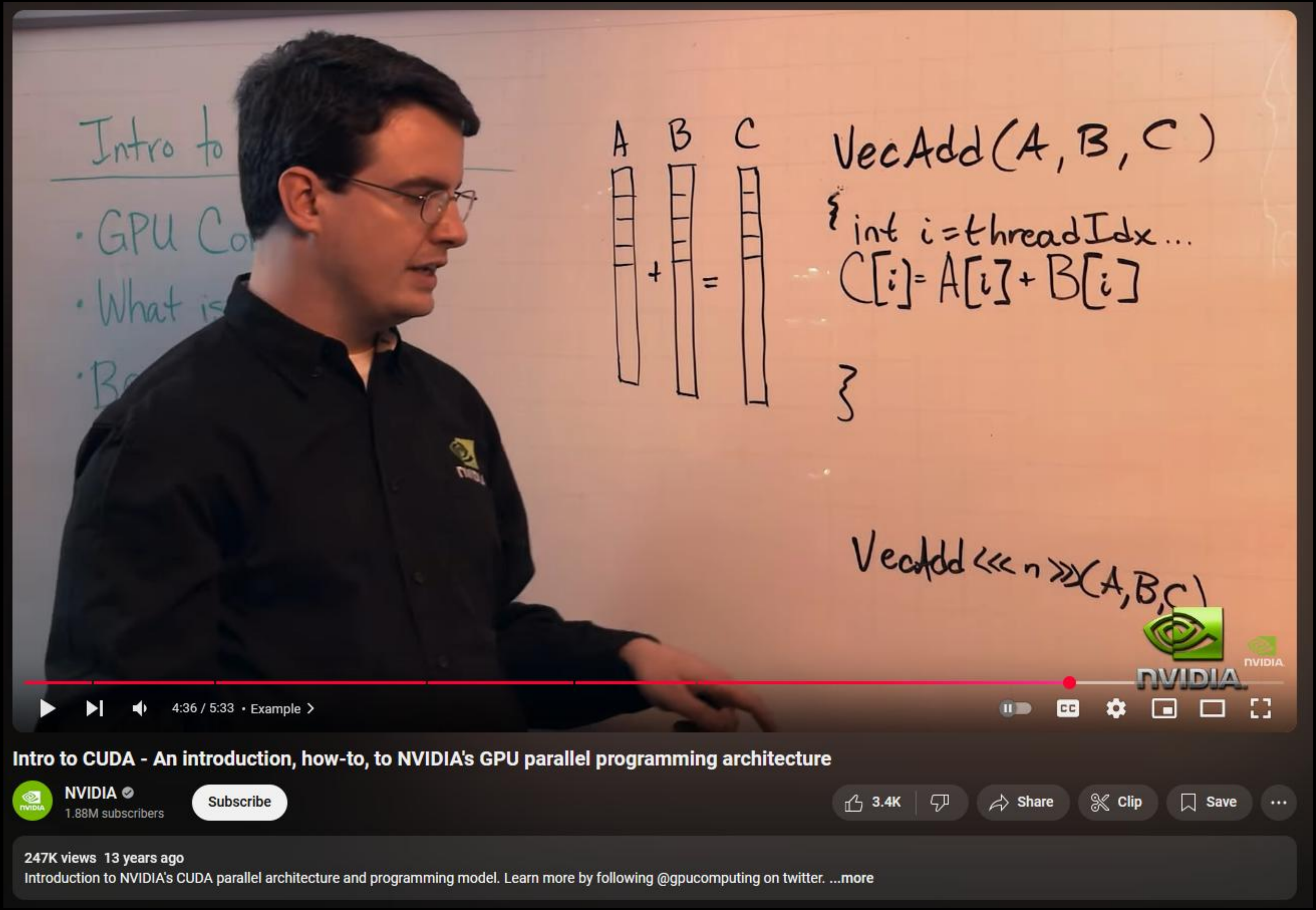
Vertex

Optimal utilization
In unified architecture

Heavy Geometry
Workload Perf = 11

Heavy Pixel
Workload Perf = 11

- *An introduction to Modern GPU Architecture,* Ashu Rege



```
VecAdd(A, B, C)
{
 int i=threadIdx...
C[i]= A[i] + B[i]

}

VecAdd<<<n>>>(A,B,C)
```

Intro to CUDA - An introduction, how-to, to NVIDIA's GPU parallel programming architecture

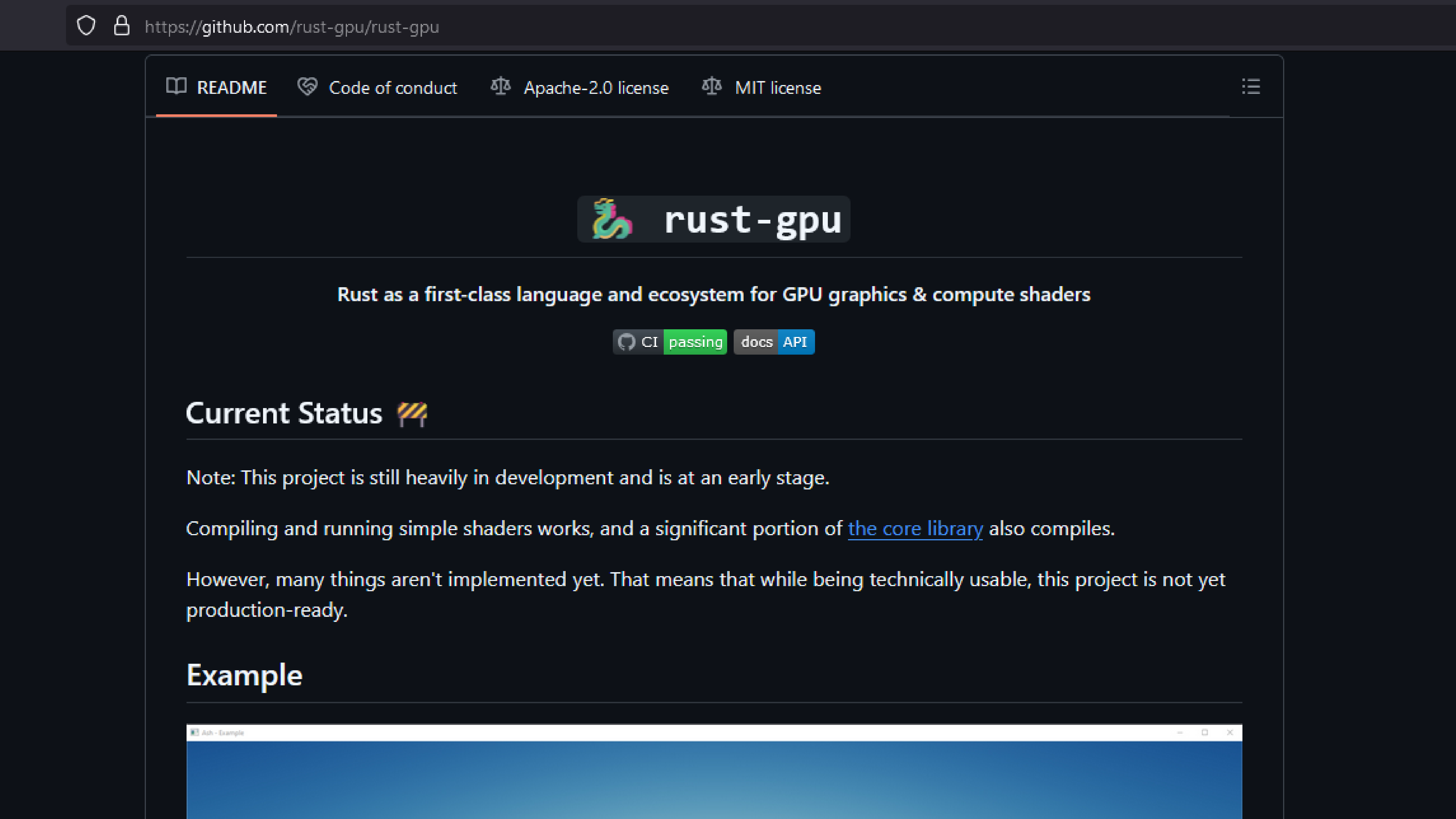
NVIDIA
1.88M subscribers

247K views 13 years ago

Introduction to NVIDIA's CUDA parallel architecture and programming model. Learn more by following @gpucomputing on twitter. ...more

Rust

Programming a GPU with it.



🐉 rust-gpu

Rust as a first-class language and ecosystem for GPU graphics & compute shaders

Current Status 🚧

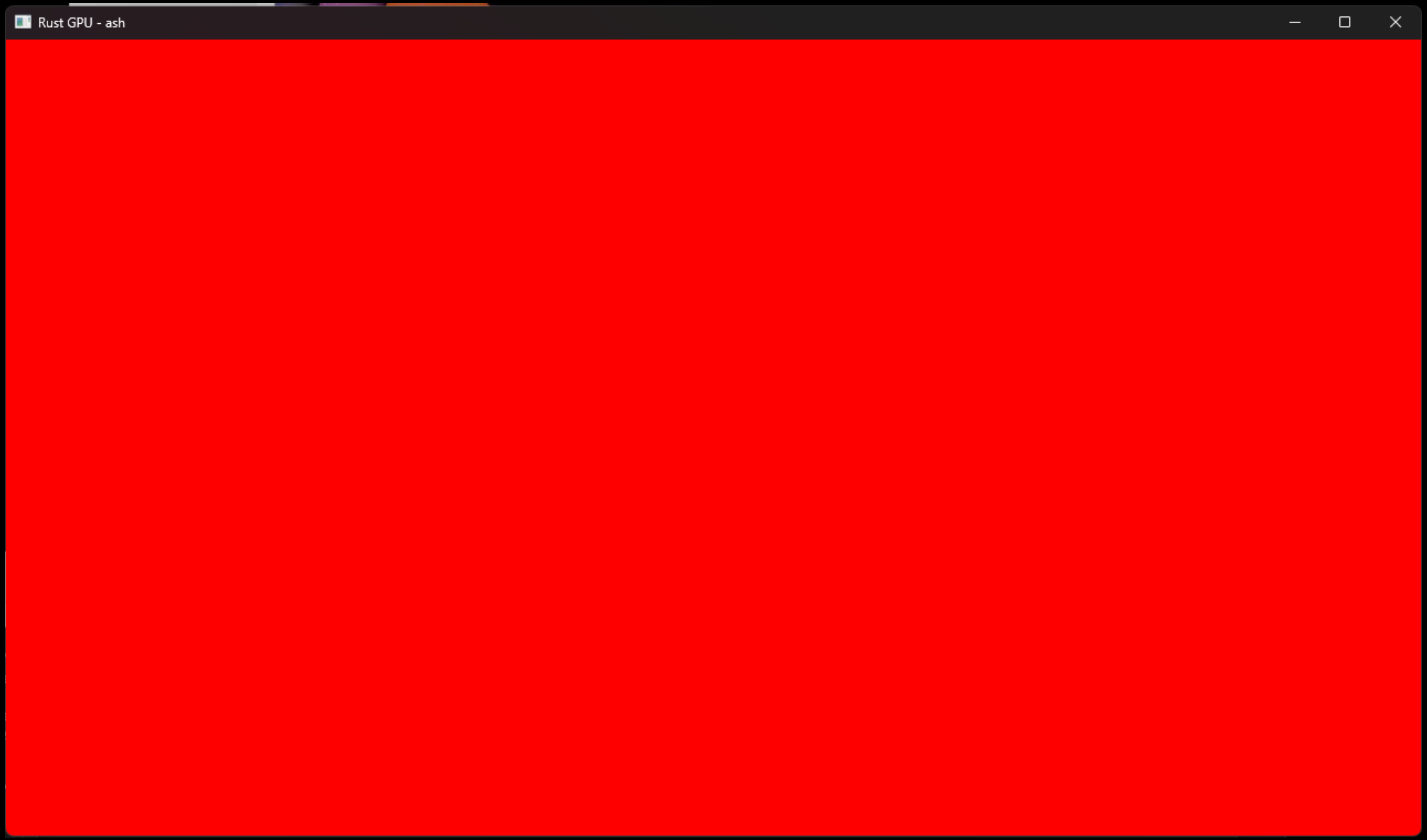
Note: This project is still heavily in development and is at an early stage.

Compiling and running simple shaders works, and a significant portion of the core library also compiles.

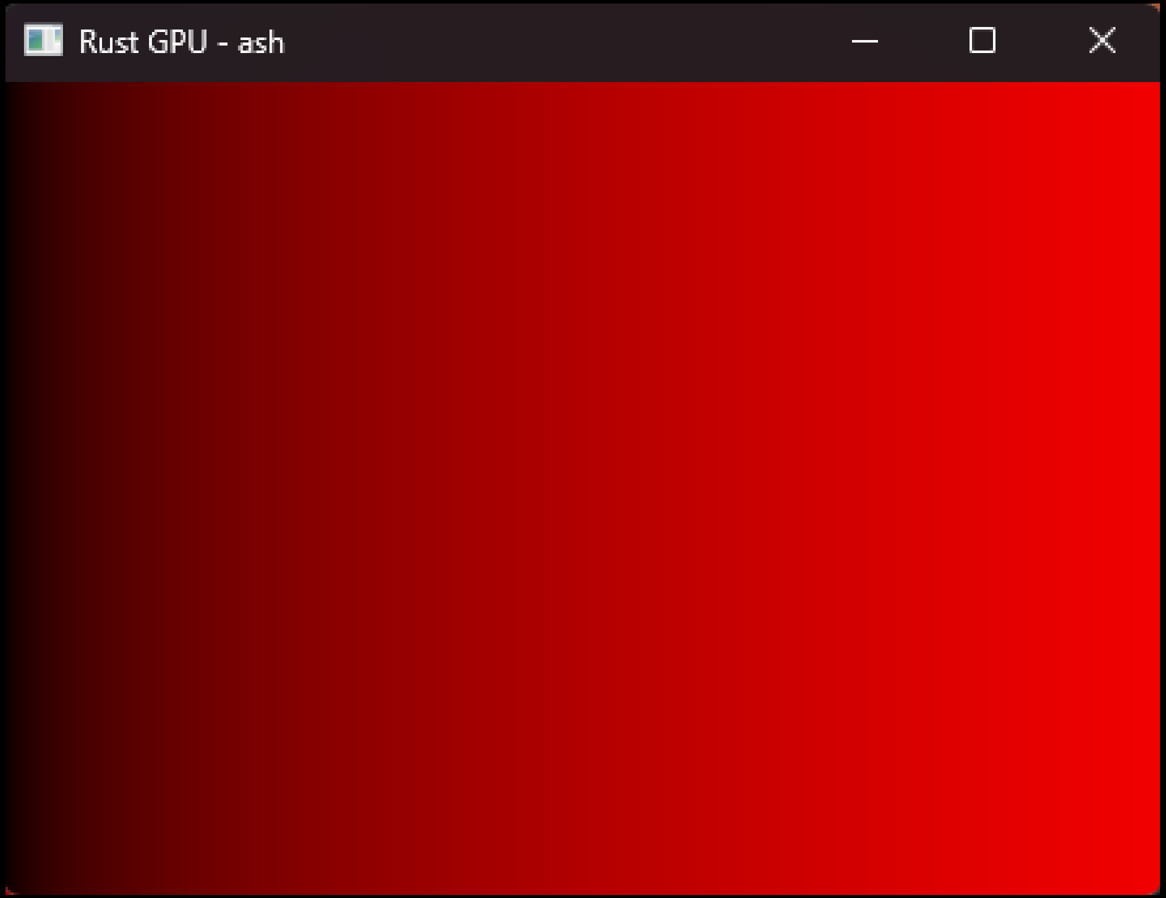
However, many things aren't implemented yet. That means that while being technically usable, this project is not yet production-ready.

Example

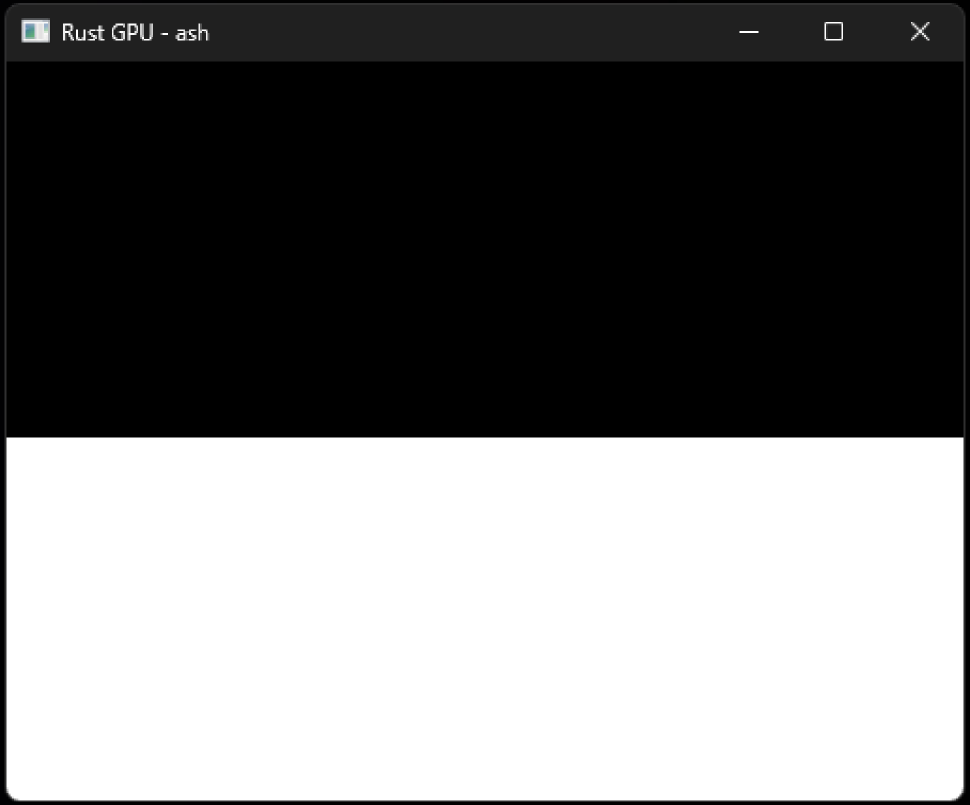
```
fn calculate_pixel(x: f32, y: f32) -> Vec3 {
    return vec3(1.0, 0.0, 0.0);
}
```



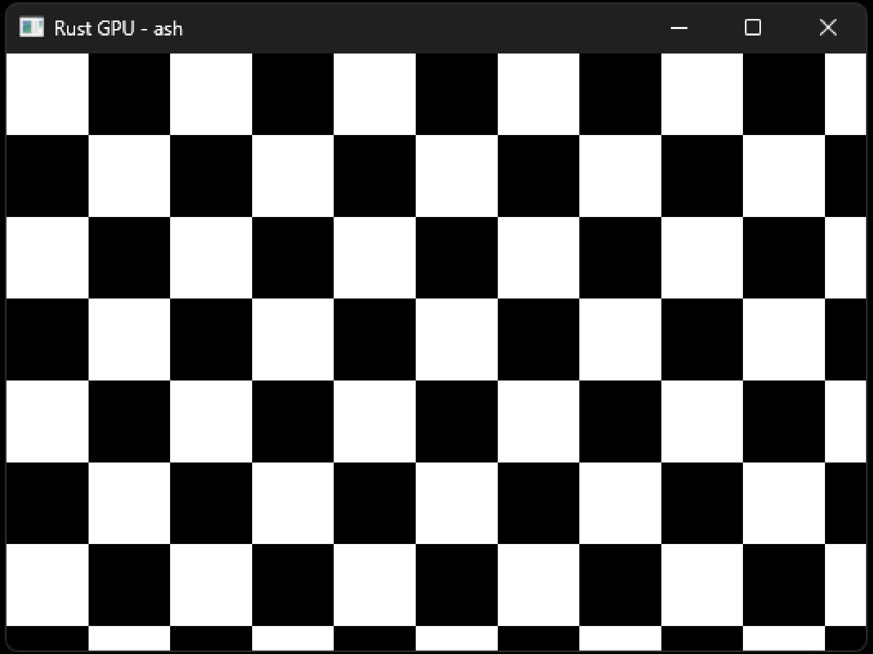
```
fn calculate_pixel(x: f32, y: f32) -> Vec3 {
    return vec3(x / 500.0, 0.0, 0.0);
}
```



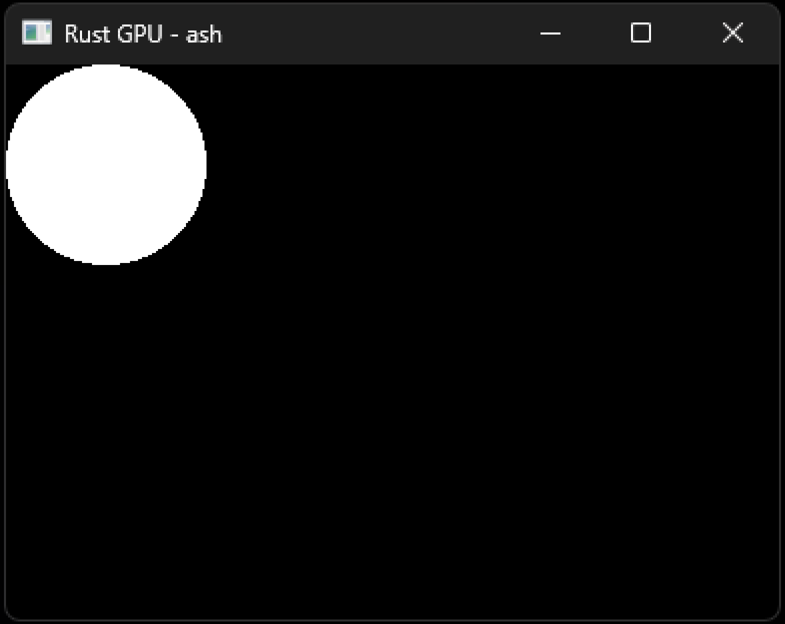
```
fn calculate_pixel(x: f32, y: f32) -> Vec3 {
    let mut intensity = 0.0;

    if y > 200.0 {
        intensity = 1.0;
    }

    return vec3(intensity, intensity, intensity);
}
```



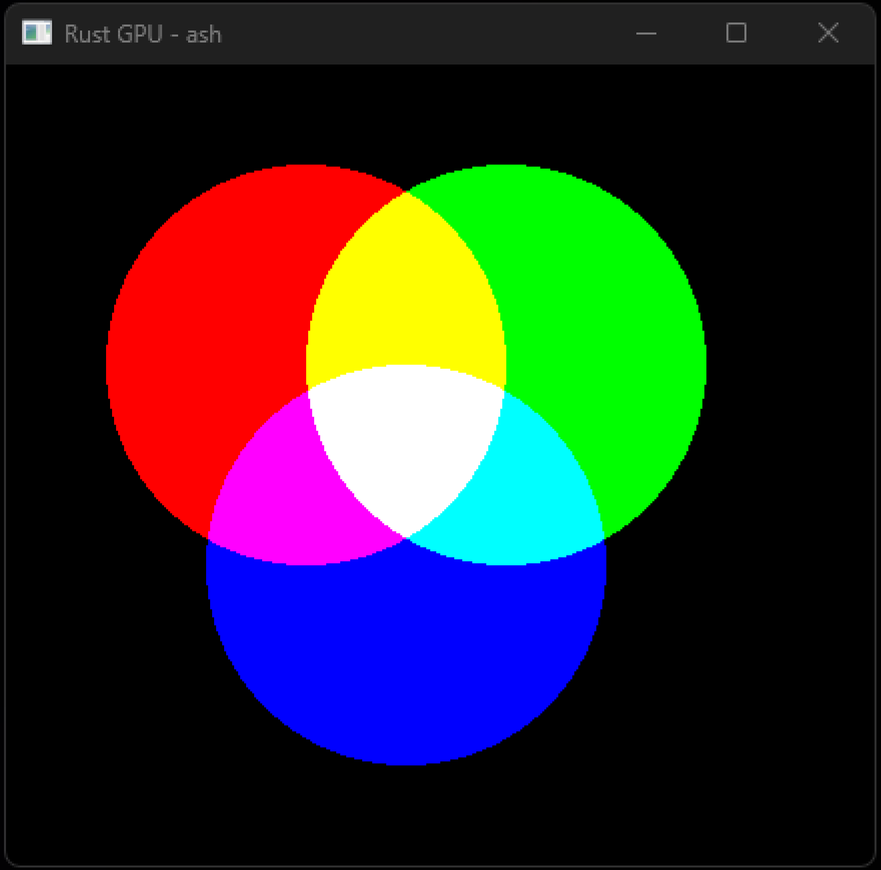
```rust
fn calculate_pixel(x: f32, y: f32) -> Vec3 {
    const GRID_SIZE: f32 = 50.0;

    let mut intensity = 0.0;

    let isOnEvenRow = (y / GRID_SIZE).floor() % 2.0 == 0.0;
    let isOnEvenCol = (x / GRID_SIZE).floor() % 2.0 == 0.0;


    let is_white_square = isOnEvenRow == isOnEvenCol;

    if is_white_square {
        intensity = 1.0;
    }

    return vec3(intensity, intensity, intensity);
}
```



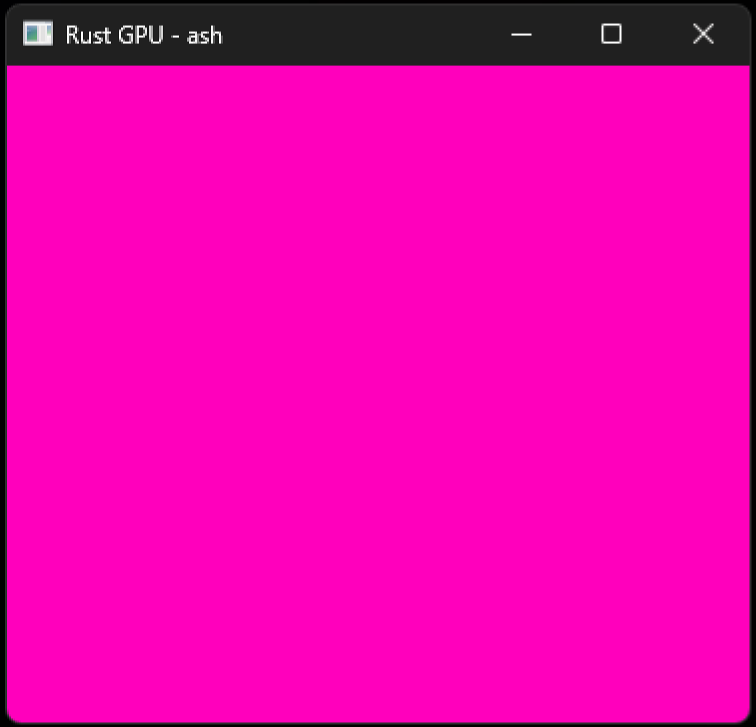
```rust
fn length(v: Vec2) -> f32 {
    let dot = v.x * v.x + v.y * v.y;
    dot.sqrt()
}

fn circle(x: f32, y: f32, size: f32, pixel_x: f32, pixel_y: f32) -> bool {
    let center = vec2(x, y);
    let distance = length(vec2(pixel_x, pixel_y) - center);
    let is_pixel_inside_circle = distance < size;
    return is_pixel_inside_circle;
}

fn calculate_pixel(x: f32, y: f32) -> Vec3 {

    let mut intensity = 0.0;

    let is_in_circle = circle(50.0, 50.0, 50.0, x, y);

    if is_in_circle {
        intensity = 1.0;
    }

    return vec3(intensity, intensity, intensity);
}
```



```
fn length(v: Vec2) -> f32 {
    let dot = v.x * v.x + v.y * v.y;
    dot.sqrt()
}

fn circle(x: f32, y: f32, size: f32, pixel_x: f32, pixel_y: f32) -> bool {
    let center = vec2(x, y);
    let distance = length(vec2(pixel_x, pixel_y) - center);
    let is_pixel_inside_circle = distance < size;
    return is_pixel_inside_circle;
}

fn calculate_pixel(x: f32, y: f32) -> Vec3 {

    let is_in_red_circle   = circle(150.0, 150.0, 100.0, x, y);
    let is_in_green_circle = circle(250.0, 150.0, 100.0, x, y);
    let is_in_blue_circle  = circle(200.0, 250.0, 100.0, x, y);

    let red   = if is_in_red_circle   { 1.0 } else { 0.0 };
    let green = if is_in_green_circle { 1.0 } else { 0.0 };
    let blue  = if is_in_blue_circle  { 1.0 } else { 0.0 };

    return vec3(red, green, blue);
}
```

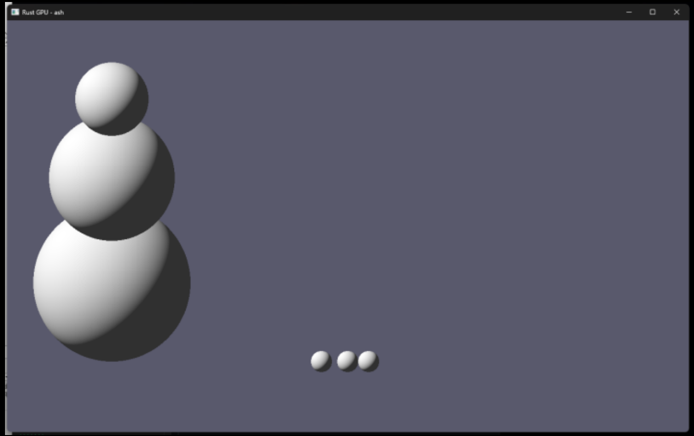


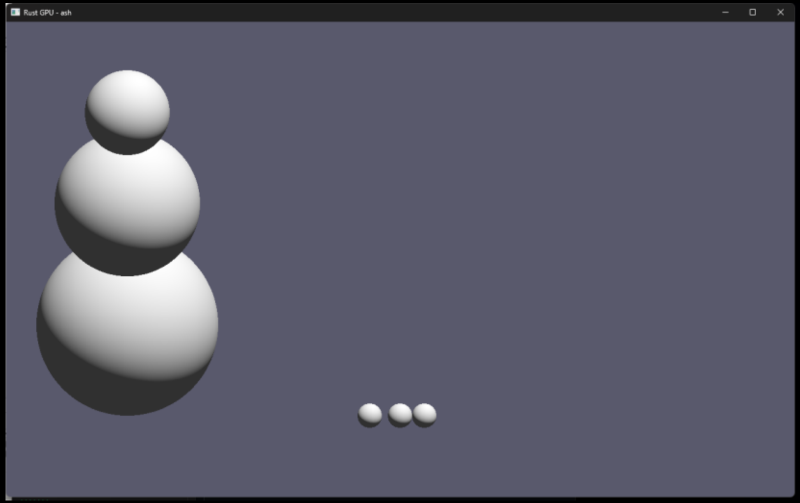
```rust
#![cfg_attr(target_arch = "spirv", no_std)]

use glam::{vec2, vec3, vec4, Vec2, Vec3, Vec4};
use spirv_std::spirv;
use shared::*;
use spirv_std::num_traits::Float;
use core::f32::consts::PI;

#[spirv(vertex)]
pub fn main_vs(
    #[spirv(vertex_index)] vert_id: i32,
    #[spirv(position)] out_pos: &mut Vec4,
) {
    // Create a "full screen triangle" by mapping the vertex index.
    // ported from https://www.saschawillems.de/blog/2016/08/13/vulkan-tutorial-on-rendering-a-fullscreen-quad-without-buffers/
    let uv = vec2(((vert_id << 1) & 2) as f32, (vert_id & 2) as f32);
    let pos = 2.0 * uv - Vec2::ONE;

    *out_pos = pos.extend(0.0).extend(1.0);
}

#[spirv(fragment)]
pub fn main_fs(
    #[spirv(frag_coord)] in_frag_coord: Vec4,
    #[spirv(push_constant)] constants: &ShaderConstants,
    output: &mut Vec4
) {
    let color = calculate_pixel(in_frag_coord.x, in_frag_coord.y);

    *output = vec4(color.x, color.y, color.z, 1.0);
}


fn calculate_pixel(x: f32, y: f32) -> Vec3 {
    return vec3(1.0, 0.0, 0.5);
}
```

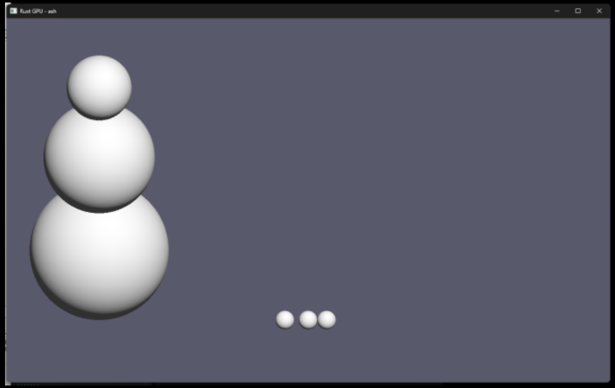


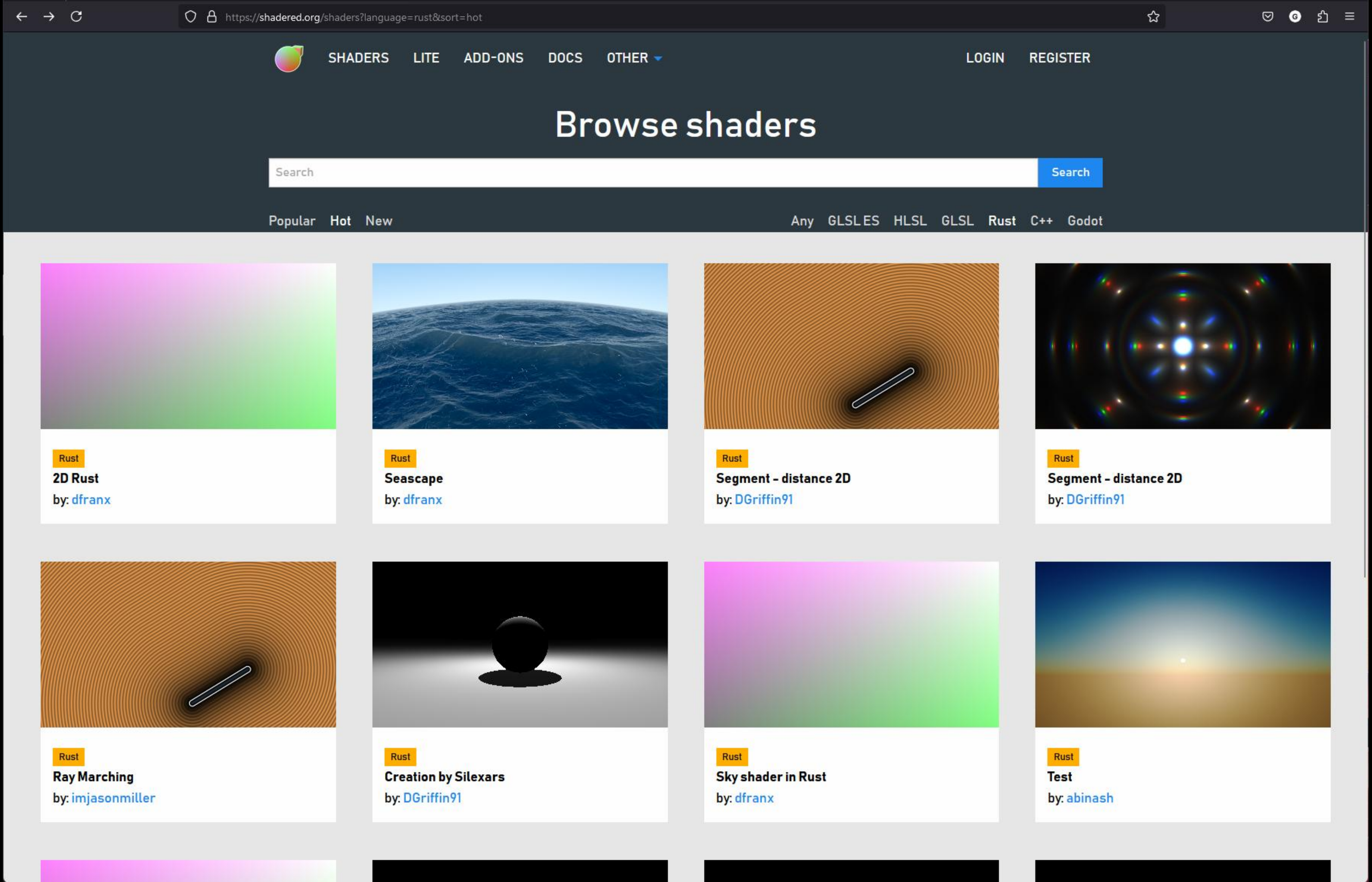
```rust
#[spirv(fragment)]
pub fn main_fs(
    #[spirv(frag_coord)] in_frag_coord: Vec4,
    #[spirv(push_constant)] constants: &ShaderConstants,
    output: &mut Vec4
) {

    let sun_dir = vec3(
        constants.cursor_x / (constants.width as f32) - 0.5,
        constants.cursor_y / (constants.height as f32) - 0.5,
        0.3).normalize();


    let pixel_loc = vec2(in_frag_coord.x, in_frag_coord.y);


    let spheres = [
        vec4(200.0, 500.0, 0.0, 150.0),
        vec4(200.0, 300.0, 0.0, 120.0),
        vec4(200.0, 150.0, 0.0, 70.0),

        vec4(600.0, 650.0, 0.0, 20.0),
        vec4(650.0, 650.0, 0.0, 20.0),
        vec4(690.0, 650.0, 0.0, 20.0),
    ];


    let mut pixel_color = vec4(0.1, 0.1, 0.15, 1.0);

    for i in 0..spheres.len() {
        let sphere_data = spheres[i];
        let sphere_col = sphere(sphere_data.x, sphere_data.y, sphere_data.w, pixel_loc.x, pixel_loc.y, sun_dir);
        if sphere_col.w > 0.1 {
            pixel_color = sphere_col;
        }
    }


    *output = pixel_color;


}
```

```rust
fn sphere(x: f32, y: f32, size: f32, pixel_x: f32, pixel_y: f32, light_dir: Vec3) -> Vec4 {
    let center = vec2(x, y);
    let distance = length(vec2(pixel_x, pixel_y) - center);
    if distance < size {


        let local_pos = vec2(pixel_x - x, pixel_y - y);
        let local_pos = vec3(local_pos.x, local_pos.y, (size*size - distance*distance).sqrt());
        let local_pos = local_pos.normalize();


        let light_intensity = local_pos.dot(light_dir);

        let light_intensity = light_intensity.max(0.03);

        return vec4(light_intensity, light_intensity, light_intensity, 1.0);

    }
    vec4(0.0, 0.0, 0.0, 0.0)
}
```









https://shadered.org/shaders?language=rust&sort=hot

Browse shaders

Rust
2D Rust
by: dfranx

Rust
Seascape
by: dfranx

Rust
Segment - distance 2D
by: DGriffin91

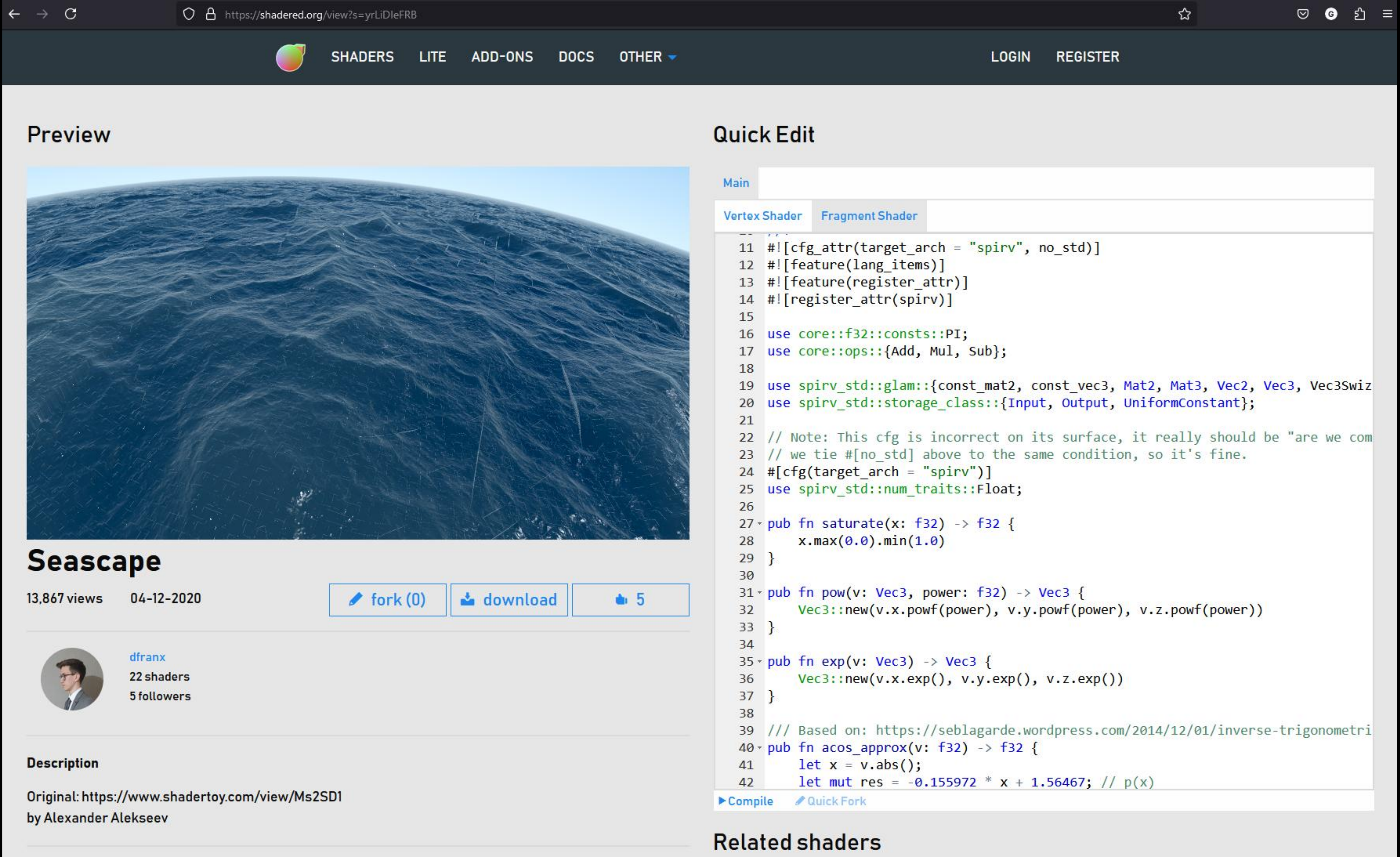
Rust
Segment - distance 2D
by: DGriffin91

Rust
Ray Marching
by: imjasonmiller

Rust
Creation by Silexars
by: DGriffin91

Rust
Sky shader in Rust
by: dfranx

Rust
Test
by: abinash



Preview

Seascape

13,867 views 04-12-2020

dfranx
22 shaders
5 followers

Description

Original: https://www.shadertoy.com/view/Ms2SD1
by Alexander Alekseev

Quick Edit

```rust
#![cfg_attr(target_arch = "spirv", no_std)]
#![feature(lang_items)]
#![feature(register_attr)]
#![register_attr(spirv)]

use core::f32::consts::PI;
use core::ops::{Add, Mul, Sub};

use spirv_std::glam::{const_mat2, const_vec3, Mat2, Mat3, Vec2, Vec3, Vec3Swiz
use spirv_std::storage_class::{Input, Output, UniformConstant};

// Note: This cfg is incorrect on its surface, it really should be "are we com
// we tie #[no_std] above to the same condition, so it's fine.
#[cfg(target_arch = "spirv")]
use spirv_std::num_traits::Float;

pub fn saturate(x: f32) -> f32 {
    x.max(0.0).min(1.0)
}

pub fn pow(v: Vec3, power: f32) -> Vec3 {
    Vec3::new(v.x.powf(power), v.y.powf(power), v.z.powf(power))
}

pub fn exp(v: Vec3) -> Vec3 {
    Vec3::new(v.x.exp(), v.y.exp(), v.z.exp())
}

/// Based on: https://seblagarde.wordpress.com/2014/12/01/inverse-trigonometri
pub fn acos_approx(v: f32) -> f32 {
    let x = v.abs();
    let mut res = -0.155972 * x + 1.56467; // p(x)
```

Related shaders

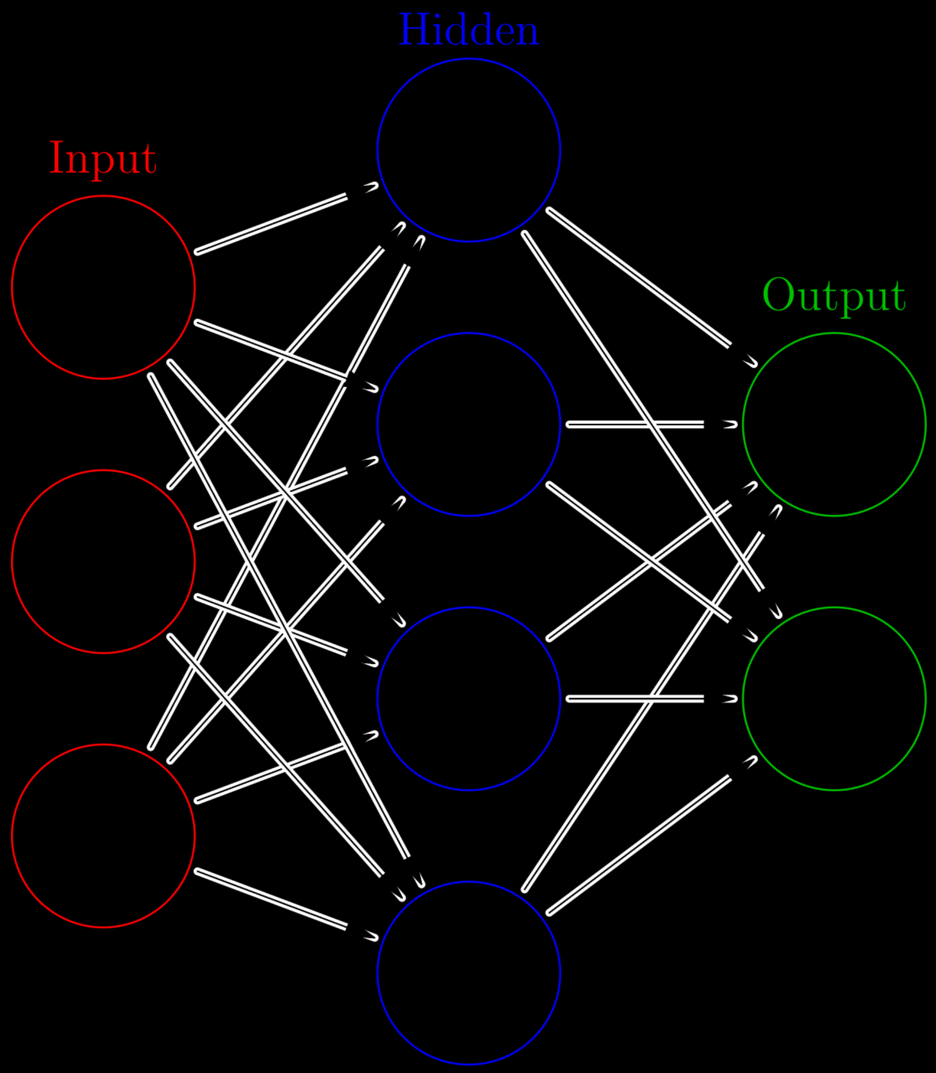
Compute

Graphics is fun, but what about crunching numbers?

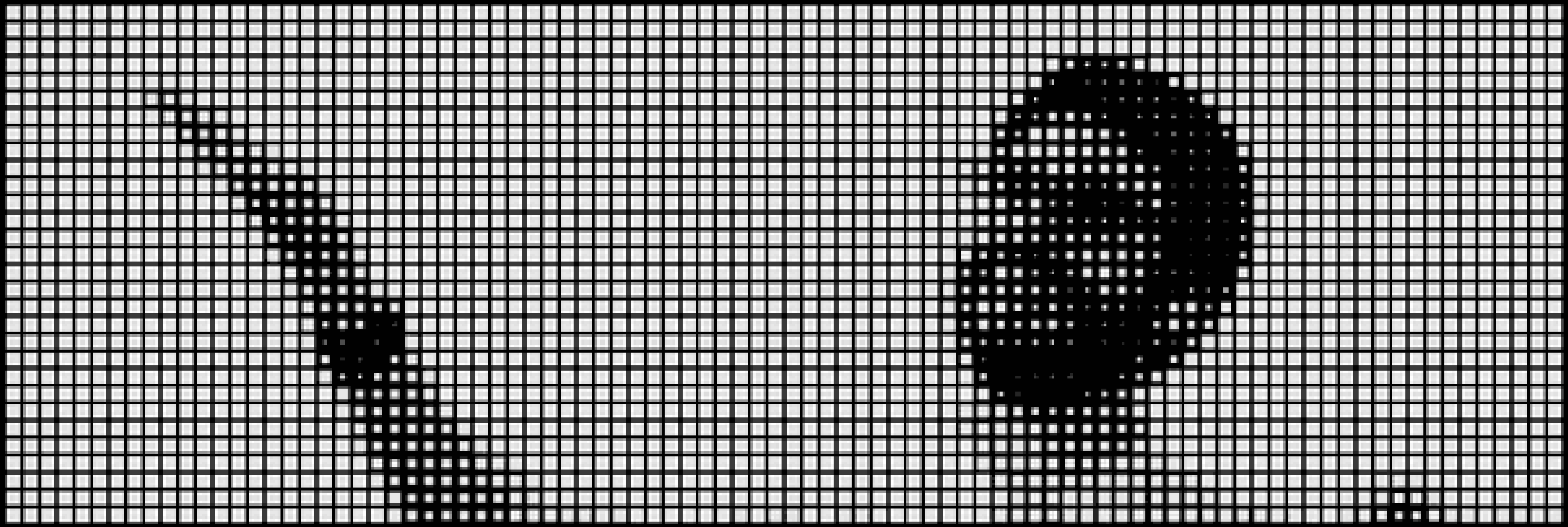
```rust
#![cfg_attr(target_arch = "spirv", no_std)]
// HACK(eddyb) can't easily see warnings otherwise from `spirv-builder` builds.
#![deny(warnings)]

use glam::UVec3;
use spirv_std::{glam, spirv};

// Adapted from the wgpu hello-compute example

pub fn collatz(mut n: u32) -> Option<u32> {
    let mut i = 0;
    if n == 0 {
        return None;
    }
    while n != 1 {
        n = if n % 2 == 0 {
            n / 2
        } else {
            // Overflow? (i.e. 3*n + 1 > 0xffff_ffff)
            if n >= 0x5555_5555 {
                return None;
            }
            // TODO: Use this instead when/if checked add/mul can work: n.checked_mul(3)?.checked_add(1)?
            3 * n + 1
        };
        i += 1;
    }
    Some(i)
}

// LocalSize/numthreads of (x = 64, y = 1, z = 1)
#[spirv(compute(threads(64)))]
pub fn main_cs(
    #[spirv(global_invocation_id)] id: UVec3,
    #[spirv(storage_buffer, descriptor_set = 0, binding = 0)] prime_indices: &mut [u32],
) {
    let index = id.x as usize;
    prime_indices[index] = collatz(prime_indices[index]).unwrap_or(u32::MAX);
}
```

Why?

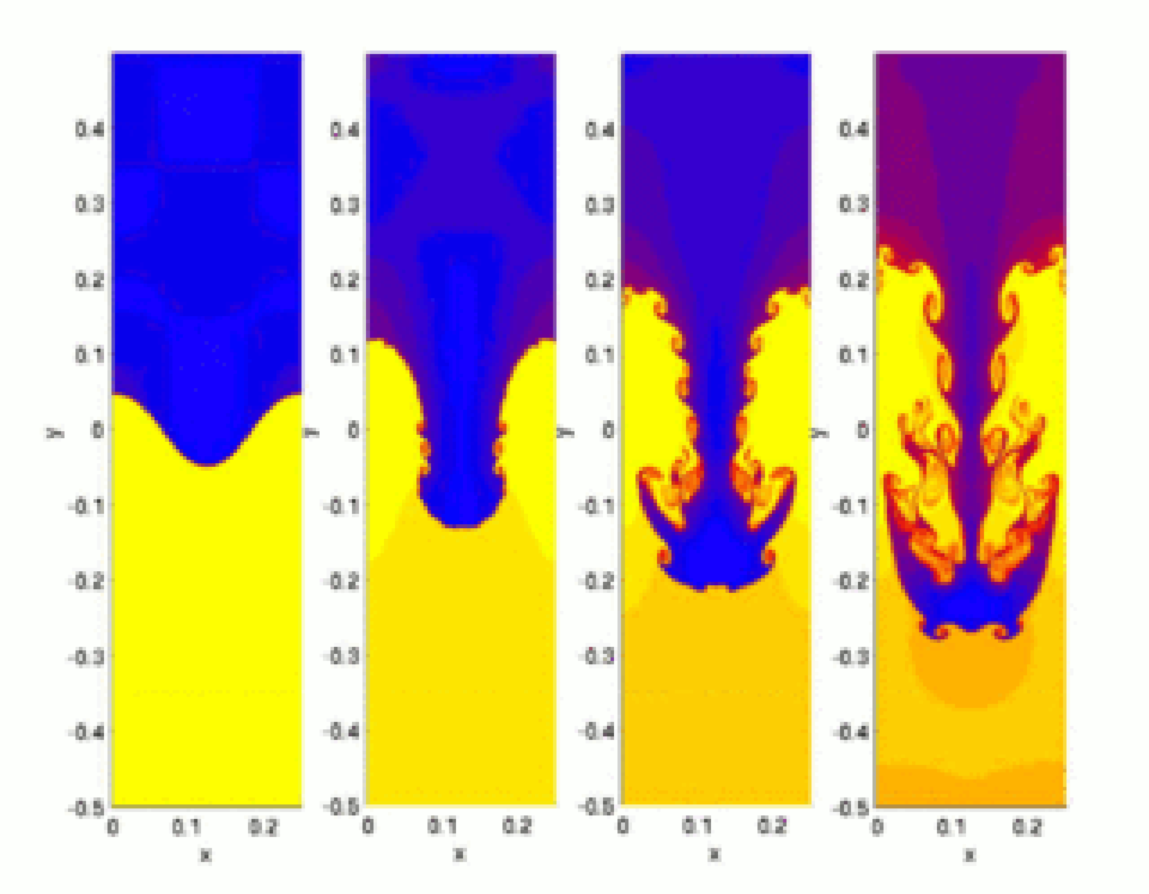
Use cases for Rust GPU



- Neural Network, Wikipedia



- Video, Danial Shiffman, Processing.org



- Fluid Dynamics, Wikipedia

- Computer Graphics, Wikipedia

many more…

Questions?