Real-Time Fluid Dynamics Simulation in Julia

Using `Waterlily.jl`, `GLMakie.jl`

References:

- Real-Time Fluid Dynamics Simulation in Julia using Waterlily.jl, GLMakie.jl
- Tutorial 08x13

Table of Contents

Real-Time Fluid Dynamics Simulation in Julia

```julia
begin
    using PlutoUI
    PlutoUI.TableOfContents(indent=true, depth=4, aside=true)
end
```

```julia
using GLMakie
```

```julia
using WaterLily
```

```
record_volume (generic function with 1 method)
```

```julia
function record_volume(sim, data_func;
    name="file.mp4",
    duration=1,
    step=0.1,
    size=1024
)
    ## Set up visualization data and figure (GLMakie)
    data = data_func(sim) |> d -> Observable(d)
    set_theme!(
        theme_dark(),
        resolution=(size, size)
    )
    fig = volume(
        data,
        colorrange = (pi, 3pi),
        algorithm = :absorption
    )

    ## Run simulation and update figure data
    t₀ = round(sim_time(sim))
    t = range(t₀, t₀ + duration; step)

    record(fig, name, t) do tᵢ
        sim_step!(sim, tᵢ) # from WaterLily
        data[] = data_func(sim)
        println(
            "Simulation ",
            round(Int, (tᵢ - t₀) / duration * 100),
            "% complete"
        )
    end
    sim, fig
end
```

TGV

```
TGV(p, Re)
```

Taylor-Green Vortex animation function, where the two input parameters are:

- `p` the pressure and
- `Re` the Reynolds number

```julia
"""
    TGV(p, Re)

Taylor-Green Vortex animation function, where the two input parameters are:
- `p` the pressure and
- `Re` the Reynolds number
"""
function TGV(p=6, Re=100_000)
    ## Define vortex size, velocity, viscosity
    L = 2^p
    U = 1.
    ν = U * L / Re

    ## Taylor-Green Vortex initial velocity field
    function uλ(ix, vₓ)
        ## scaled coordinates
        x, y, z = @. (vₓ - 1.5) * π / L
        form = U * sin(x) * cos(y) * cos(z)
        ix == 1 && return -1. * form # u_x
        ix == 2 && return form # u_y
        return 0.0 # u_z
    end

    ## Initialize simulation
    Simulation(
        (L + 2, L + 2, L + 2), zeros(3), L;
        U, uλ, ν
    )
end
```

```
omega_mag_data (generic function with 1 method)
```

```julia
function omega_mag_data(sim)
    ## Plot the vorticity modulus
    @inside sim.flow.σ[I] = WaterLily.ω_mag(I, sim.flow.u) * sim.L / sim.U
    @view sim.flow.σ[2:end-1, 2:end-1, 2:end-1]
end
```

```julia
## generate animation - turn off

# sim, fig = record_volume(
#   TGV(),
#   omega_mag_data;
#     name = "TGV.mp4",
#     duration = 20,
#     step = 0.025
# )
```

```
PlutoUI.LocalResource("TGV.mp4", :width => 500, :autoplay => "", :loop => false)
```