Fish by MLS-MPM(Moving-Least Squares Material Point Method) of Taichi

10 Feb. 2026

CAD の obj ファイルを読み込んで，MPM 計算し，VTU に保存するようにしてみた．とりあえず，魚はアクティブには動かない．MPM は材料で区別していて，物体毎は id 化されていない．後日検討．
材料は，雪（snow），砂（sand），弾性体（elastic）の三つが登録されている．適当な位置から落としてみた．

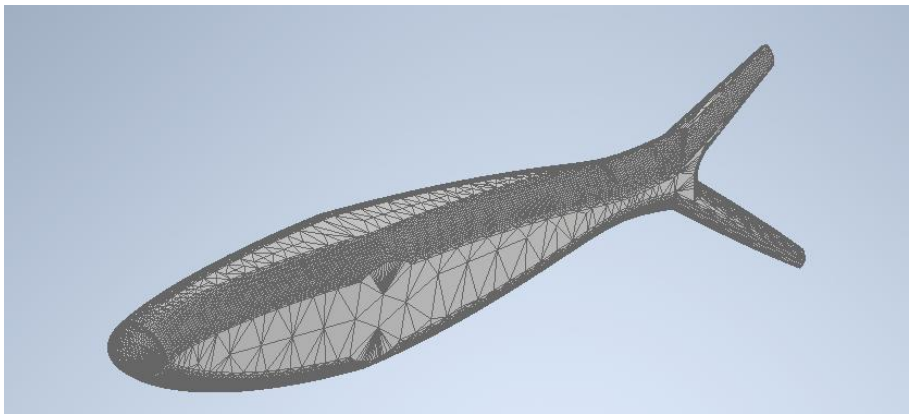
(1) インストール

以下を参考に，mlsmpm がインストールされていることが前提．
https://www.kikulab.chibatech.ac.jp/wordpress/wp-content/uploads/2026/02/MPM.pdf

このファイル軍の以下を書き換えた．

engine/mpm_solver.py	# CAD の obj ファイルから粒子を生成できるようにした
VTU.py	# paraview 用に VTU ファイルを吐き出すようにした．

(2) 魚もモデル
　いつもの魚
　fishbody.obj



(3) ファイルの作成
. mpm_solver.py
trimesh を使って obj ファイルを読み込み，点群化し，その中をボクセル化し，それを粒子化した．粒子は 50%重なっている．となりの粒子の表面上に対象の粒子の中心がある．特に意味は無し．

. . . .

```
    # ******** for CAD model ************
    def add_CADmodel(self, file_name, offset, scale,
                     material,
                     color=0xFFFFFF,
                     sample_density=None,
                     velocity=None):
```

```
        mesh = tm.load(file_name)                   # load CAD obj file
        mesh.apply_scale(scale)                     # scaling
        mesh.vertices += offset                     # translational move

        is_success = tm.repair.fill_holes(mesh)     # hole check and repair
        voxelized_mesh = mesh.voxelized(pitch=self.dx/2).fill()     # vozelized the obj and fill the
inside using scale of dx/2
        voxelized_points_np = voxelized_mesh.points.astype(np.float32)     # chage the tpye of
particle point [x, y, z] [m] to np float32
        num_new_particles = voxelized_points_np.shape[0]    # total particle number

        assert self.n_particles[None] + num_new_particles <= self.max_num_particles

        self.set_source_velocity(velocity=velocity)     # self.source_velocity[None][i] = [0,0,0] initial
velocity
        self._seed_from_ndarray(voxelized_points_np, int(self.n_particles[None]), material, color)

        self.n_particles[None] += num_new_particles

    @ti.kernel
    def _seed_from_ndarray(self,
                           points: ti.types.ndarray(),   # shape (N, dim), dtype=float32
                           start_idx: ti.i32,
                           material: ti.i32,
                           color: ti.i32 ):
        n = points.shape[0]
        for j in range(n):
            i = start_idx + j
            x = ti.Vector.zero(ti.f32, self.dim)
            for k in ti.static(range(self.dim)):
                x[k] = points[j, k]
            self.seed_particle(i, x, material, color, self.source_velocity[None], None)
```

. . .

. VTU.py

paraview 用.
meshio というパッケージが便利とのこととで使ってみた.

```
> pip install meshio
```

応力はテンソルと von misses を chaty に作ってもらった．モデルに id がないため，材料が同じ魚だと独立したパラメータの設定ができない．後日検討

```
  @ti.kernel
  def compute_stress_and_vm(self, F_field: ti.template(),     # pass the Taichi matrix field
                            mu: ti.f32,
                            lam: ti.f32,
                            stress_out: ti.types.ndarray(),     # (n,9) float32
                            vm_out: ti.types.ndarray(),         # (n,) float32
                            n: ti.i32):
    for i in range(n):
        F = F_field[i]                                  # assume F is 3x3 taichi matrix field
        J = ti.max( ti.abs(F.determinant()), 1e-8 )                     # avoid negative jitter
        B = F @ F.transpose()
        I = ti.Matrix([[1.0,0.0,0.0],[0.0,1.0,0.0],[0.0,0.0,1.0]])
        tau = mu * (B - I) + lam * ti.log(J) * I
        sigma = tau / J

        stress_out[i, 0] = sigma[0,0]
        stress_out[i, 1] = sigma[0,1]
```

```
            stress_out[i, 2] = sigma[0,2]
            stress_out[i, 3] = sigma[1,0]
            stress_out[i, 4] = sigma[1,1]
            stress_out[i, 5] = sigma[1,2]
            stress_out[i, 6] = sigma[2,0]
            stress_out[i, 7] = sigma[2,1]
            stress_out[i, 8] = sigma[2,2]

            mean = (sigma[0,0] + sigma[1,1] + sigma[2,2]) / 3.0
            sxx = sigma[0,0] - mean
            syy = sigma[1,1] - mean
            szz = sigma[2,2] - mean
            sxy = 0.5 * (sigma[0,1] + sigma[1,0])
            sxz = 0.5 * (sigma[0,2] + sigma[2,0])
            syz = 0.5 * (sigma[1,2] + sigma[2,1])
            vm_out[i] = ti.sqrt(1.5 * (sxx*sxx + syy*syy + szz*szz + 2.0*(sxy*sxy + sxz*sxz + syz*syz)))

    def write_vtuMPMASCII(self, filename, mpm: ti.template()):
        n = int( mpm.n_particles[None] )
        stress_np = np.zeros((n, 9), dtype=np.float32)
        vm_np = np.zeros((n,), dtype=np.float32)

        # Taichi -> NumPy
        pos = mpm.x.to_numpy()[:n]                  # (n, 3)
        vel = mpm.v.to_numpy()[:n]                  # (n, 3)
        mat = mpm.material.to_numpy()[:n]       # (n,)

        mu_ = mpm.mu_0
        lambda_ = mpm.lambda_0

        self.compute_stress_and_vm(mpm.F, mu_, lambda_, stress_np, vm_np, n)

        # VTU は vertex cell を使う
        cells = [("vertex", np.arange(n).reshape(-1, 1))]

        point_data = {
            "velocity": vel,          # [vx, vy, vz]
            "material": mat,          # scalar
            "von_mises": vm_np,
            "stress": stress_np,      # (n,9)
        }

        mesh = meshio.Mesh(
            points=pos,
            cells=cells,
            point_data=point_data
        )

        mesh.write(filename)
```

. mlsmpm04.py
実行ファイル

```
import taichi as ti
import numpy as np
import utils
import os
import glob
from moviepy import ImageSequenceClip
```

```
from engine.mpm_solver import MPMSolver          # ここで上記 solver を呼び出している
from VTU import VTUclass                         # VTU クラス

write_to_disk = False

# Parameters
SimulationTime = 1.0                             # シミュレーション時間
dt = 0.002                                       # いまは関係ない
fileName = "fishbody.obj"                        # CAD ファイル名


# Try to run on GPU
# ti.init(arch=ti.gpu, device_memory_GB=4.0)

ti.init(arch=ti.cpu)                             # cpu で計算

mpm = MPMSolver(res=(64, 64, 64), size=5, max_num_particles=2 ** 15, use_ggui=True) #  粒子生成

#  以下魚三匹．それぞれ材料を変更してみた．黄色が砂．白が雪．赤が弾性体
mpm.add_CADmodel(fileName, offset=[4, 2, 3], scale=[20.0, 20.0, 20.0],
material=MPMSolver.material_snow)
mpm.add_CADmodel(fileName, offset=[4, 5, 4], scale=[20.0, 20.0, 20.0],
material=MPMSolver.material_elastic)
mpm.add_CADmodel(fileName, offset=[4, 8, 5], scale=[20.0, 20.0, 20.0],
material=MPMSolver.material_sand)

mpm.set_gravity((0, 0, -9.8))

@ti.kernel
def set_color(ti_color: ti.template(), material_color: ti.types.ndarray(), ti_material: ti.template()):
    for I in ti.grouped(ti_material):
        material_id = ti_material[I]
        color_4d = ti.Vector([0.0, 0.0, 0.0, 1.0])
        for d in ti.static(range(3)):
            color_4d[d] = material_color[material_id, d]
        ti_color[I] = color_4d

res = (1024, 760)
window = ti.ui.Window("Real MPM 3D", res, vsync=True)
canvas = window.get_canvas()
scene = ti.ui.Scene()
camera = ti.ui.make_camera()
camera.position(-4, -12, 12)
camera.lookat(2, 3, 4)
camera.up(0, 0, 1)
camera.fov(40)
particles_radius = 0.03

# ground lines
def draw_ground(scene, L=10.0, N=20):
    verts = np.zeros((4*(N+1), 3), dtype=np.float32)
    for i in range(N+1):
        x = i * L / N
        verts[4*i+0] = [x, 0, 0]
        verts[4*i+1] = [x, L, 0]
        verts[4*i+2] = [0, x, 0]
        verts[4*i+3] = [L, x, 0]
    vbo = ti.Vector.field(3, dtype=ti.f32, shape=4*(N+1))
    vbo.from_numpy(verts)
    scene.lines(vbo, width=1, color=(0.1,0.1,0.1))
```

```
def render():
    camera.track_user_inputs(window, movement_speed=0.03, hold_key=ti.ui.RMB)
    scene.set_camera(camera)

    scene.ambient_light((0, 0, 0))
    set_color(mpm.color_with_alpha, material_type_colors, mpm.material)

    scene.particles(mpm.x, per_vertex_color=mpm.color_with_alpha, radius=particles_radius)

    scene.point_light(pos=(0.5, 1.5, 0.5), color=(0.5, 0.5, 0.5))
    scene.point_light(pos=(0.5, 1.5, 1.5), color=(0.5, 0.5, 0.5))

    draw_ground(scene)   # ground z = 0 plane

    canvas.scene(scene)

def show_options():                    # 画面上でマウスを使って操作できる
    global particles_radius

    window.GUI.begin("Solver Property", 0.05, 0.1, 0.2, 0.10)
    window.GUI.text(f"Current particle number {mpm.n_particles[None]}")
    particles_radius = window.GUI.slider_float("particles radius ", particles_radius, 0, 0.1)
    window.GUI.end()

    window.GUI.begin("Camera", 0.05, 0.3, 0.3, 0.16)
    camera.curr_position[0] = window.GUI.slider_float("camera pos x", camera.curr_position[0], -10,
10)
    camera.curr_position[1] = window.GUI.slider_float("camera pos y", camera.curr_position[1], -10,
10)
    camera.curr_position[2] = window.GUI.slider_float("camera pos z", camera.curr_position[2], -10,
10)

    camera.curr_lookat[0] = window.GUI.slider_float("camera look at x", camera.curr_lookat[0], -10,
10)
    camera.curr_lookat[1] = window.GUI.slider_float("camera look at y", camera.curr_lookat[1], -10,
10)
    camera.curr_lookat[2] = window.GUI.slider_float("camera look at z", camera.curr_lookat[2], -10,
10)

    window.GUI.end()

material_type_colors = np.array([
    [0.1, 0.1, 1.0, 0.8],
    [236.0 / 255.0, 84.0 / 255.0, 59.0 / 255.0, 1.0],
    [1.0, 1.0, 1.0, 1.0],
    [1.0, 1.0, 0.0, 1.0]
]
)

os.makedirs("image", exist_ok=True)          # 画像保存用ディレクトリ作成
os.makedirs("VTU", exist_ok=True)            # VTU 保存用ディテクトリ作成

vtu = VTUclass()

t = 0.0
cnt = 0

while t<SimulationTime:
    print(f"time={t:.3f}")
```

```
        mpm.step(4e-3)                          # ソルバ．ここがホントの dt

        render()
        # show_options()
        window.save_image(f"./image/frame{cnt:04}.png")
        window.show()

        vtu.write_vtuMPMASCII(f"./VTU/SPH{cnt:04}.vtu", mpm )

        t += dt
        cnt += 1

# for movie
files = sorted(glob.glob("./image/frame*.png"))
if len(files) > 0:
    clip = ImageSequenceClip(files, fps=60)
    clip.write_videofile("out.mp4", codec="libx264", audio=False)
```

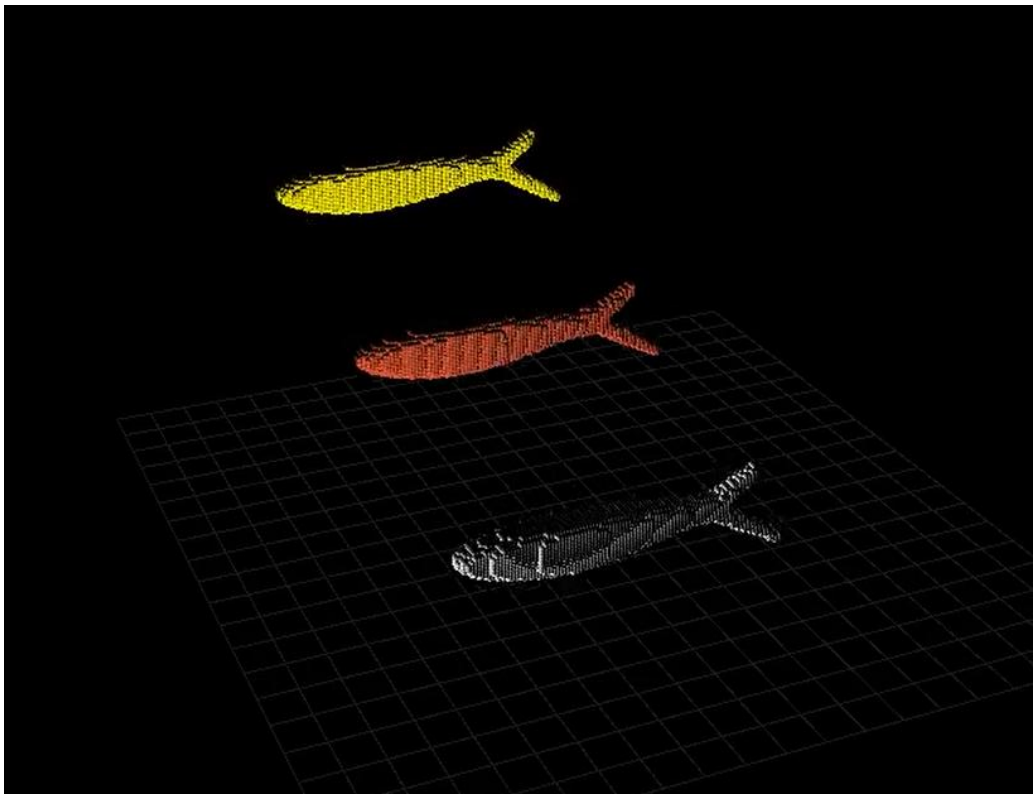
(4) 実行

taichi を activate して実行

> source taichi_env/bin/activate

> python3 mlsmpm04.py

数分で終わる．



Paraview による可視化は，別資料参照

```
print(dir(mpm))
```

```
['C', 'E', 'F', 'F_bound', 'Jp', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getstate__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_data_oriented',
'_seed_from_ndarray', 'add_CADmodel', 'add_bounding_box', 'add_cube', 'add_ellipsoid', 'add_mesh',
'add_ngon', 'add_particles', 'add_sphere_collider', 'add_surface_collider', 'add_texture_2d',
'all_time_max_velocity', 'alpha', 'block', 'block_offset', 'build_pid', 'clear_grid_postprocess',
'clear_particles', 'color', 'color_with_alpha', 'compute_max_grid_velocity', 'compute_max_velocity',
'copy_dynamic', 'copy_dynamic_nd', 'copy_ranged', 'copy_ranged_nd', 'default_dt', 'dim', 'dx', 'g2p',
'g2p2g', 'g2p2g_allowed_cfl', 'gravity', 'grid', 'grid_bounding_box', 'grid_m',
'grid_normalization_and_gravity', 'grid_postprocess', 'grid_size', 'grid_v', 'input_grid', 'inv_dx',
'lambda_0', 'last_time_final_particles', 'leaf_block_size', 'material', 'material_elastic', 'material_sand',
'material_snow', 'material_stationary', 'material_water', 'materials', 'max_num_particles', 'mu_0',
'n_particles', 'nu', 'num_grids', 'offset', 'p2g', 'p_mass', 'p_rho', 'p_vol', 'padding', 'particle', 'particle_info',
'pid', 'quant', 'random_point_in_unit_polygon', 'random_point_in_unit_sphere', 'read_restart',
'recover_from_external_array', 'res', 'sand_projection', 'seed', 'seed_ellipsoid',
'seed_from_external_array', 'seed_from_voxels', 'seed_particle', 'seed_polygon', 'set_gravity',
'set_source_velocity', 'source_bound', 'source_velocity', 'stencil_range', 'step', 'support_plasticity',
'surface_separate', 'surface_slip', 'surface_sticky', 'surfaces', 't', 'total_substeps', 'unbounded',
'use_adaptive_dt', 'use_bls', 'use_emitter_id', 'use_g2p2g', 'use_ggui',
'v',
'v_clamp_g2p2g', 'voxelizer', 'voxelizer_super_sample', 'water_density', 'write_particles',
'write_particles_ply', 'writers',
'x'
]
```