PHYSICALLY-BASED SIMULATION PROJECT MILESTONE: POSITION-BASED FLUID

Group 8

Yufeng Xiao, Mingjie Li, Sebastian Heckers

TIMELINE

week	1	2	3	4	5	6	7	8
Minimum target	Setting up framework extension for 3D pdf simulation		Collision functions with static objects		Setting up objects collision scenes			
Desired target					collision functions for simple dynamic objects		Rendering	
Bonus target							Setting up collision with sandcastel or cloth	

01.11.
Project Plan Presentation

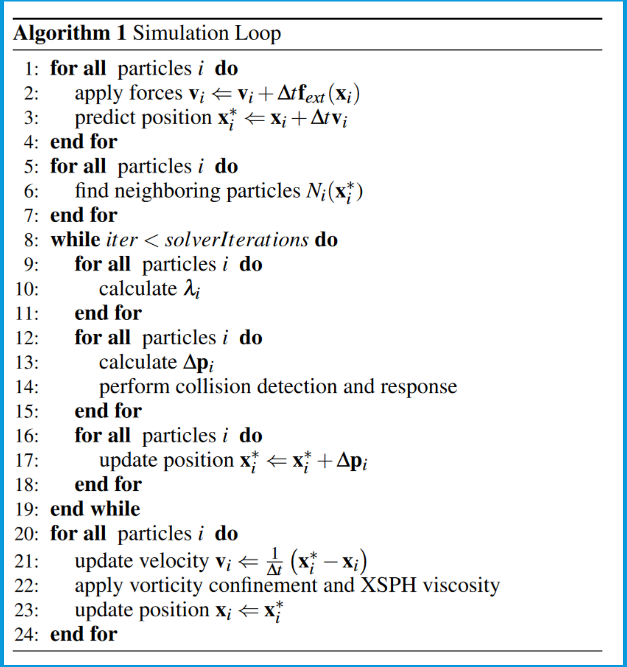
29.11.
Mid-term Presentation

23.11.
Final Presentation

MINIMAL TARGET

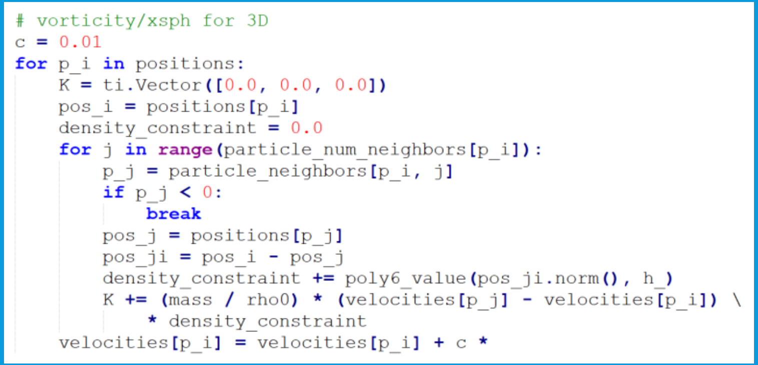
- 3D Position Based Fluid Simulation
- Collision with static convex objects (spheres, cubes…)

3D POSITION-BASED FLUID SIMULATION

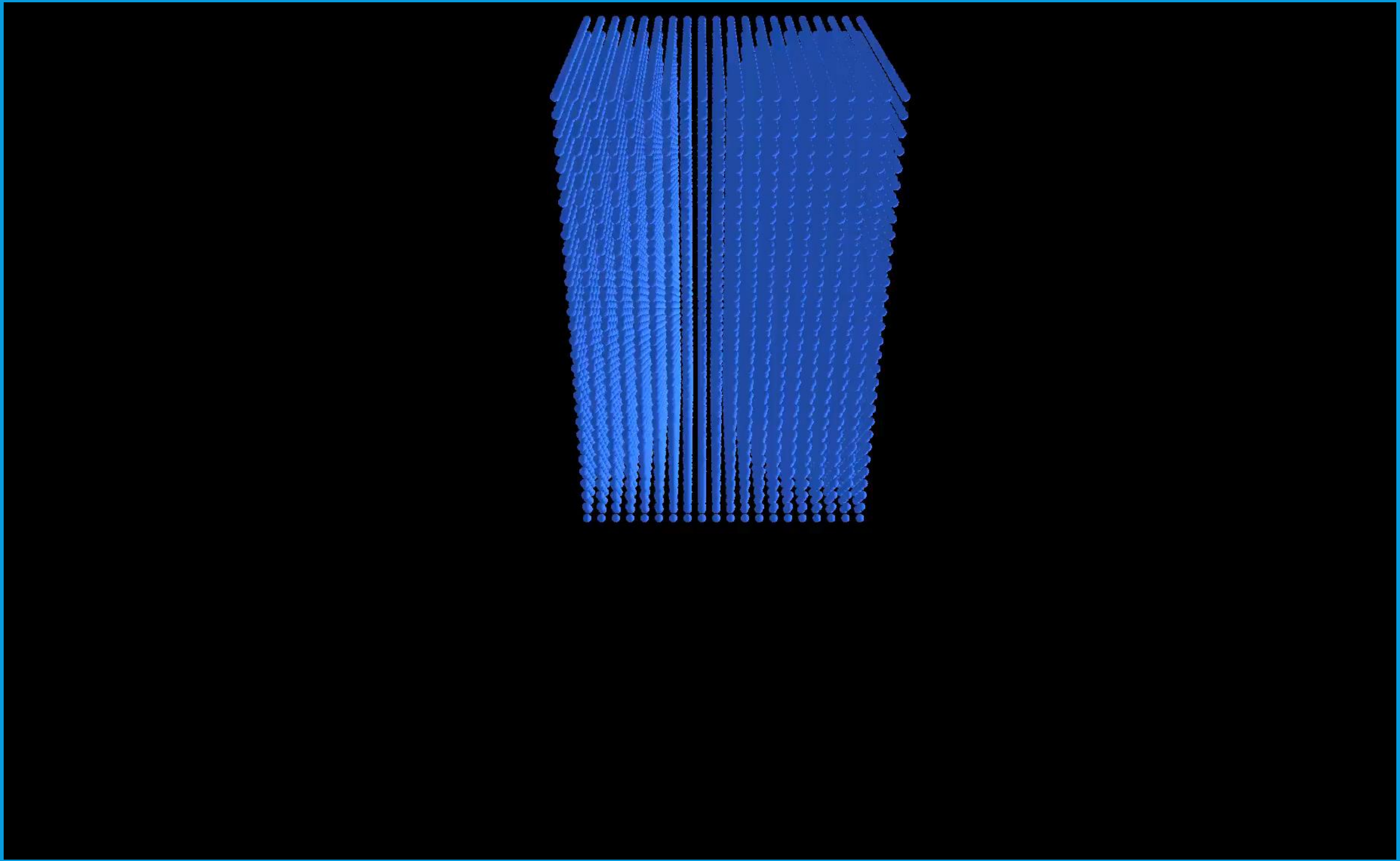


Algorithm 1 Simulation Loop

```
for all particles $i$ do
    apply forces $\mathbf{v}_i \Leftarrow \mathbf{v}_i + \Delta t \mathbf{f}_{ext}(\mathbf{x}_i)$
    predict position $\mathbf{x}_i^* \Leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
end for
for all particles $i$ do
    find neighboring particles $N_i(\mathbf{x}_i^*)$
end for
while $iter < solverIterations$ do
    for all particles $i$ do
        calculate $\lambda_i$
    end for
    for all particles $i$ do
        calculate $\Delta \mathbf{p}_i$
        perform collision detection and response
    end for
    for all particles $i$ do
        update position $\mathbf{x}_i^* \Leftarrow \mathbf{x}_i^* + \Delta \mathbf{p}_i$
    end for
end while
for all particles $i$ do
    update velocity $\mathbf{v}_i \Leftarrow \frac{1}{\Delta t}(\mathbf{x}_i^* - \mathbf{x}_i)$
    apply vorticity confinement and XSPH viscosity
    update position $\mathbf{x}_i \Leftarrow \mathbf{x}_i^*$
end for
```



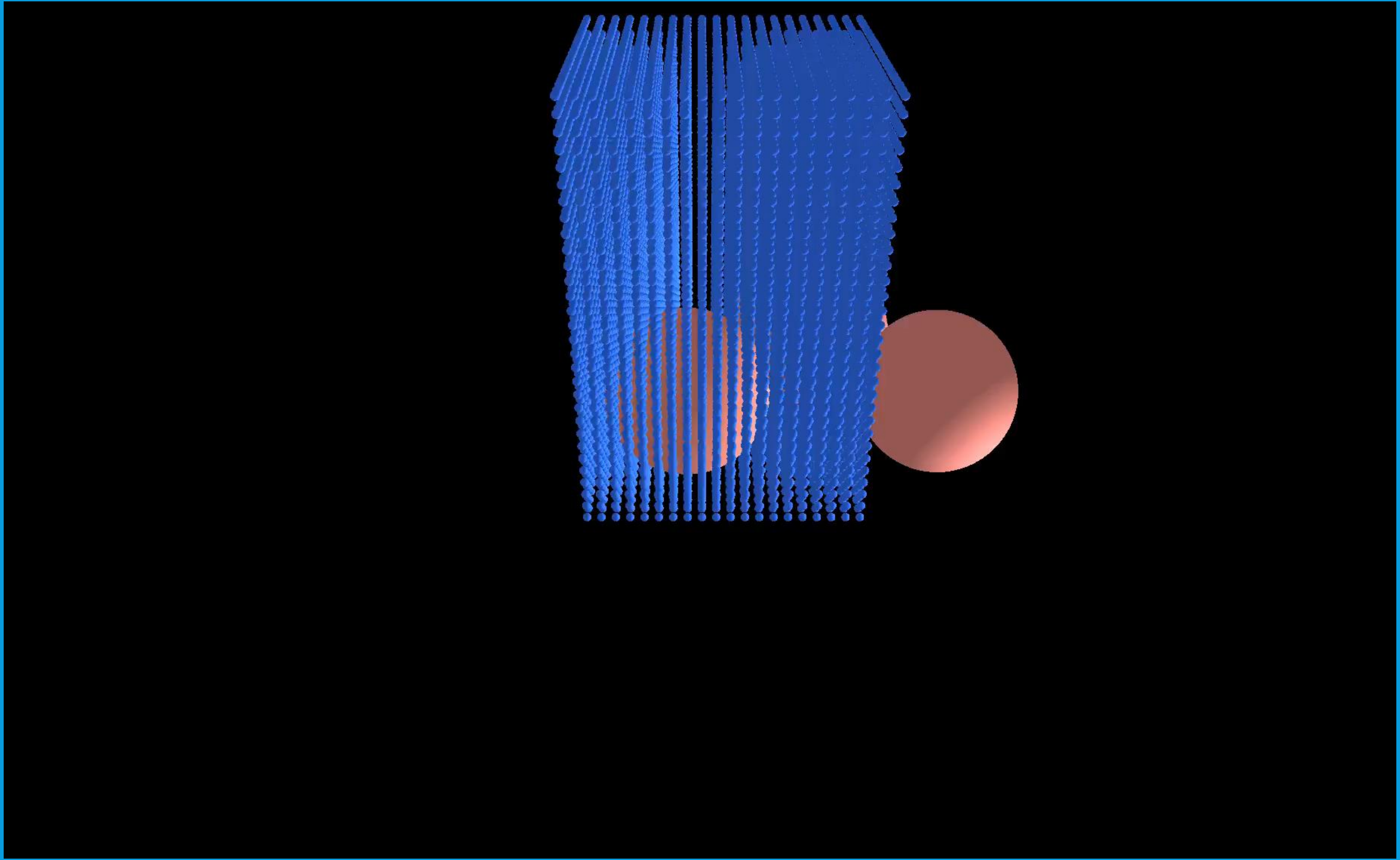
```python
# vorticity/xsph for 3D
c = 0.01
for p_i in positions:
    K = ti.Vector([0.0, 0.0, 0.0])
    pos_i = positions[p_i]
    density_constraint = 0.0
    for j in range(particle_num_neighbors[p_i]):
        p_j = particle_neighbors[p_i, j]
        if p_j < 0:
            break
        pos_j = positions[p_j]
        pos_ji = pos_i - pos_j
        density_constraint += poly6_value(pos_ji.norm(), h_)
        K += (mass / rho0) * (velocities[p_j] - velocities[p_i]) \
            * density_constraint
    velocities[p_i] = velocities[p_i] + c *
```



COLLISION WITH STATIC OBJECTS

Sphere Collision

```python
@ti.func
def particle_collide_collision_sphere(p,v):
    for i in range(num_collision_spheres):
        sdf_value = (p-collision_sphere_positions[i]).norm()- \
                    (collision_sphere_radius+particle_radius_in_world+collision_contact_offset)
        if sdf_value <= 0.:
            sdf_normal = (p-collision_sphere_positions[i])/(p-collision_sphere_positions[i]).norm()
            closest_p_on_sphere = p - sdf_value*sdf_normal
            p = closest_p_on_sphere + sdf_normal * (particle_radius_in_world \
                + collision_contact_offset + epsilon * ti.random())
            v -= v.dot(sdf_normal)*sdf_normal*2.0
            v *= collision_velocity_damping
    return p,v
```



COLLISION WITH STATIC OBJECTS

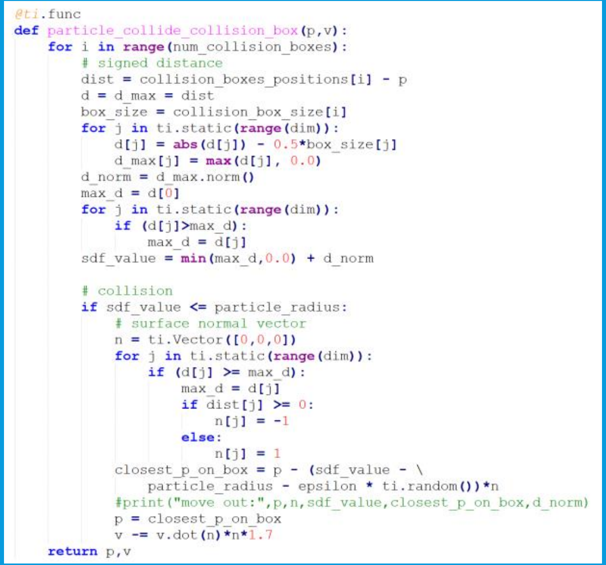
Box Collision

$$d = \begin{pmatrix} |x - x_{box}| - 0.5 * w \\ |y - y_{box}| - 0.5 * h \\ |z - z_{box}| - 0.5 * b \end{pmatrix}$$

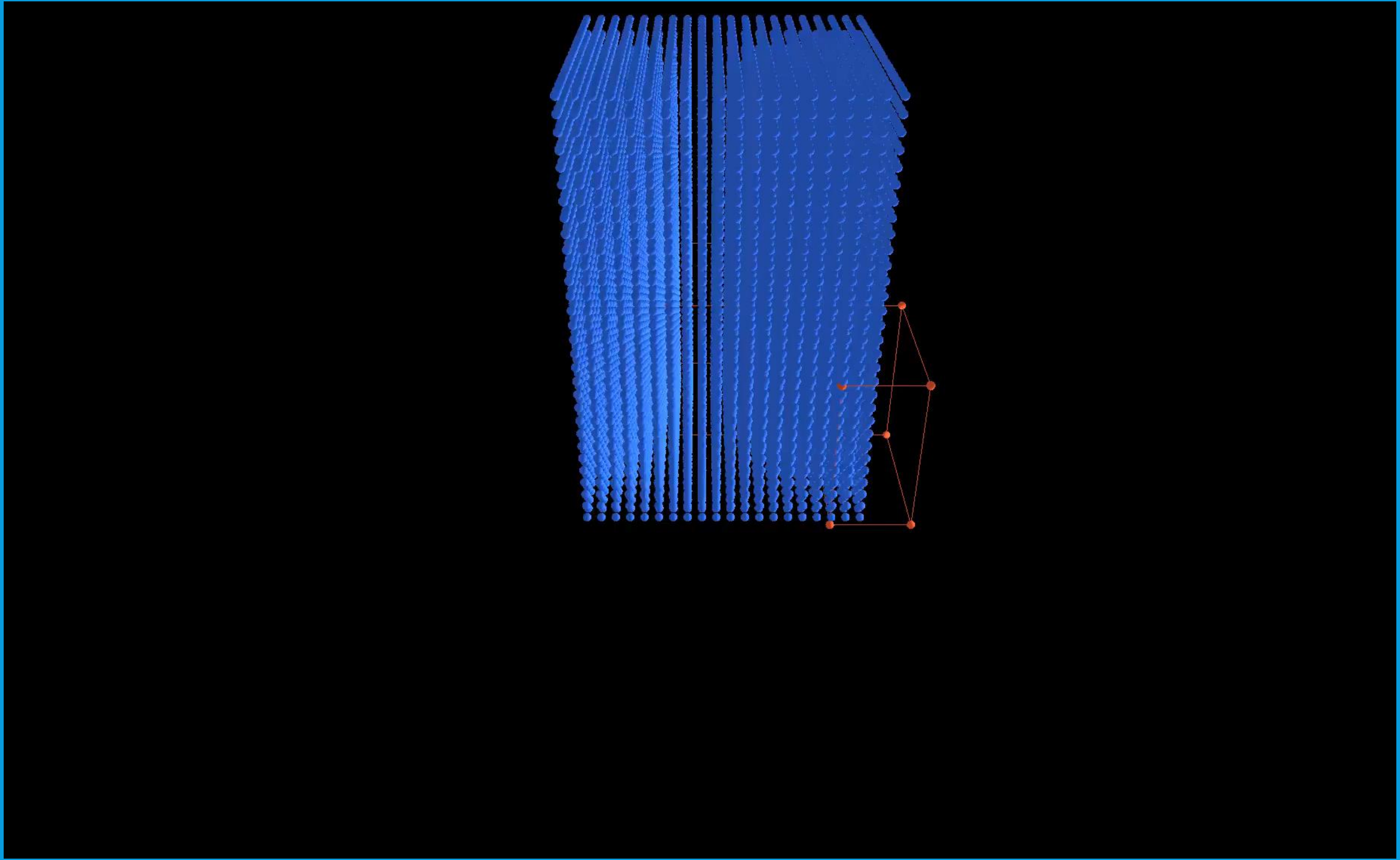
$$\Phi(x) = \min\left(\max\left(d_x, d_y, d_z\right), 0.0\right) + \left\| \begin{pmatrix} \max\left(d_x, 0.0\right) \\ \max\left(d_y, 0.0\right) \\ \max\left(d_z, 0.0\right) \end{pmatrix} \right\|$$

$$s = x - \Phi(x)n$$

▪



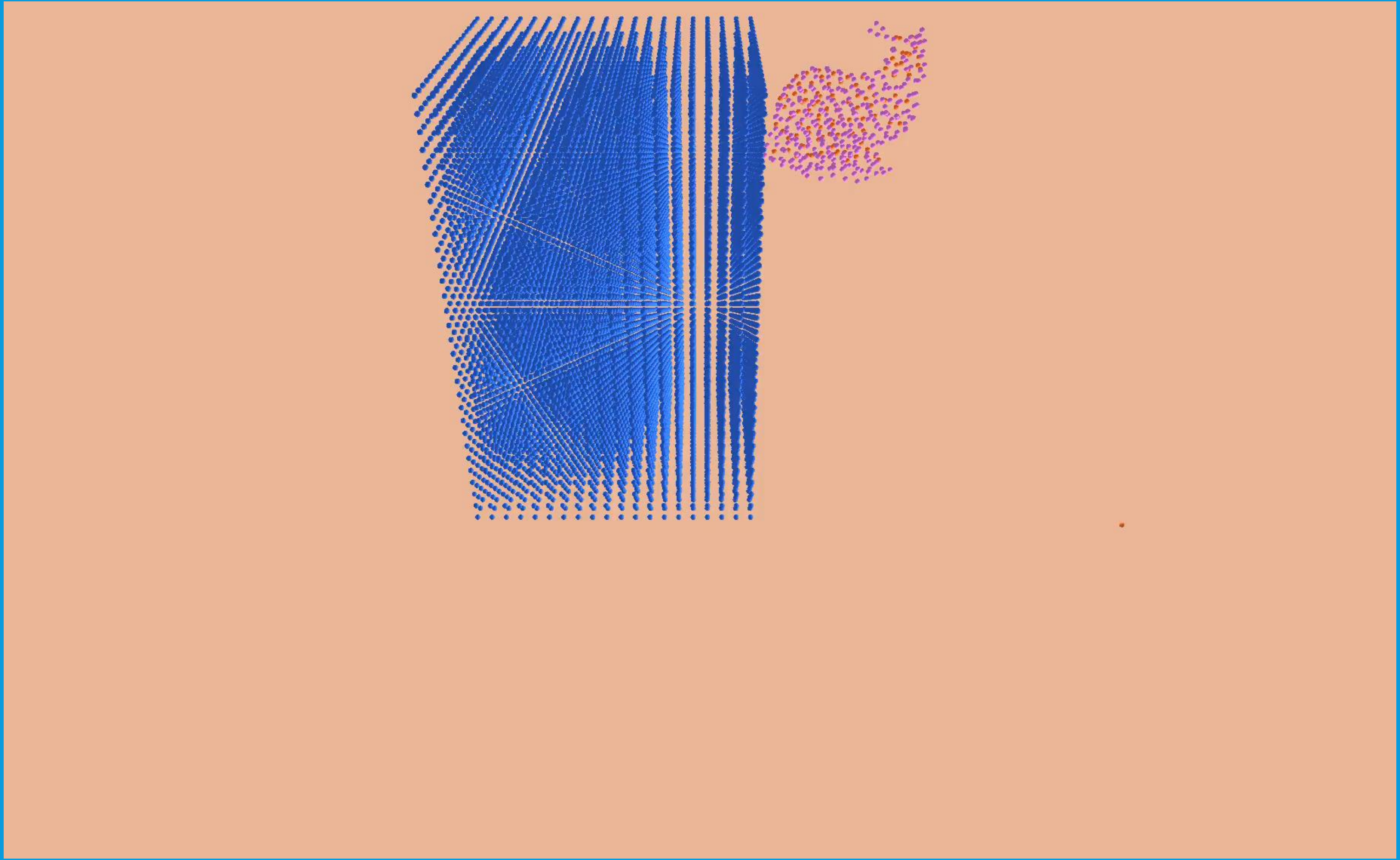
```python
@ti.func
def particle_collide_collision_box(p,v):
    for i in range(num_collision_boxes):
        # signed distance
        dist = collision_boxes_positions[i] - p
        d = d_max = dist
        box_size = collision_box_size[i]
        for j in ti.static(range(dim)):
            d[j] = abs(d[j]) - 0.5*box_size[j]
            d_max[j] = max(d[j], 0.0)
        d_norm = d_max.norm()
        max_d = d[0]
        for j in ti.static(range(dim)):
            if (d[j]>max_d):
                max_d = d[j]
        sdf_value = min(max_d,0.0) + d_norm

        # collision
        if sdf_value <= particle_radius:
            # surface normal vector
            n = ti.Vector([0,0,0])
            for j in ti.static(range(dim)):
                if (d[j] >= max_d):
                    max_d = d[j]
                    if dist[j] >= 0:
                        n[j] = -1
                    else:
                        n[j] = 1
            closest_p_on_box = p - (sdf_value - \
                particle_radius - epsilon * ti.random())*n
            #print("move out:",p,n,sdf_value,closest_p_on_box,d_norm)
            p = closest_p_on_box
            v -= v.dot(n)*n*1.7
    return p,v
```



COLLISION WITH STATIC OBJECTS

Mesh Collision

- Particles representation for non-convex rigid body
- Using meshio and SPlishHSPlasH to generate input files



MINIMAL TARGET

- 3D Position Based Fluid Simulation ✓
- Collision with static convex objects (spheres, cubes…) ✓

DESIRED TARGET

- Collision with dynamic simple objects and maybe static meshes
- Properly modeled scene
- Proper visualisation

BONUS TARGET

- Collision with sandcastle or cloth

MILESTONES

1. Setting up framework, extension for 3D PBF simulation (1-2 week) ✓
2. Collision functions with static objects (3-4 week) ✓
3. Setting up objects collision scenes and collision functions for simple dynamic objects (5-6 week)
4. Rendering and eventually bonus target (7-8 week)