Tutorial 13

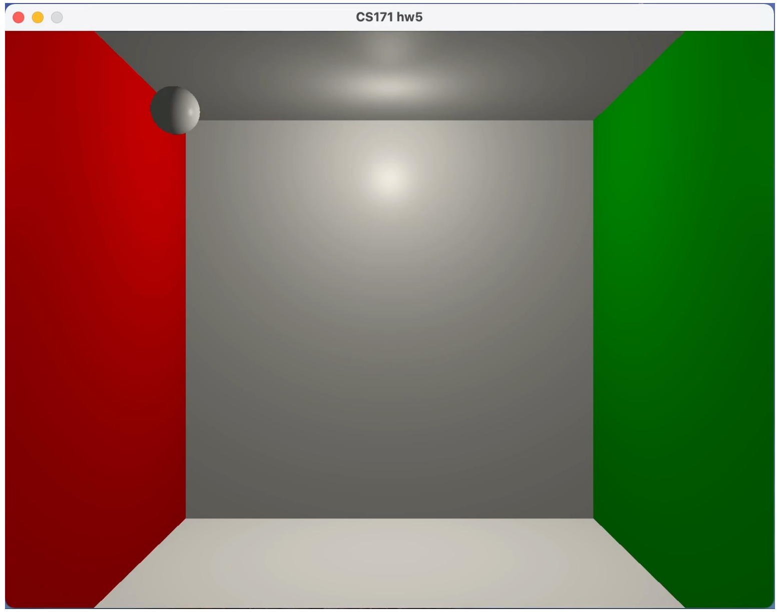
TA: Mengyun Liu, Hongtu Xu

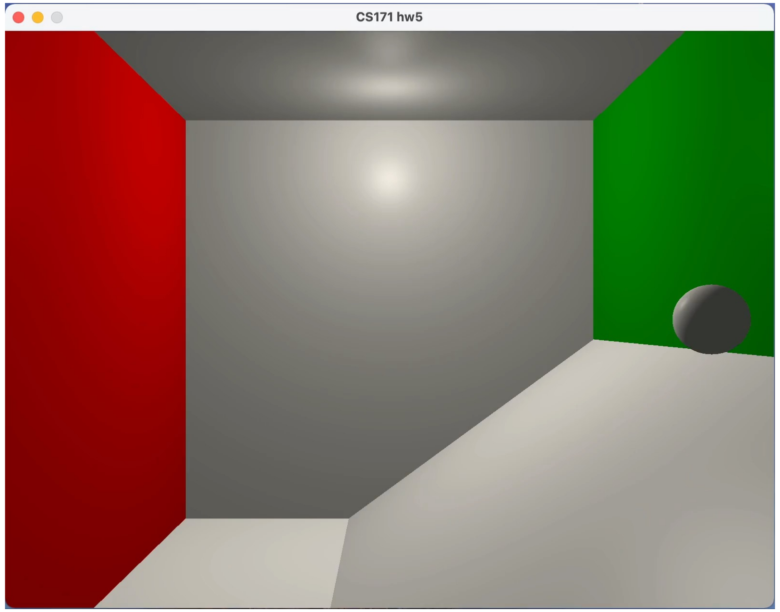
🎉 Homework 5: The Last Coding Homework :D

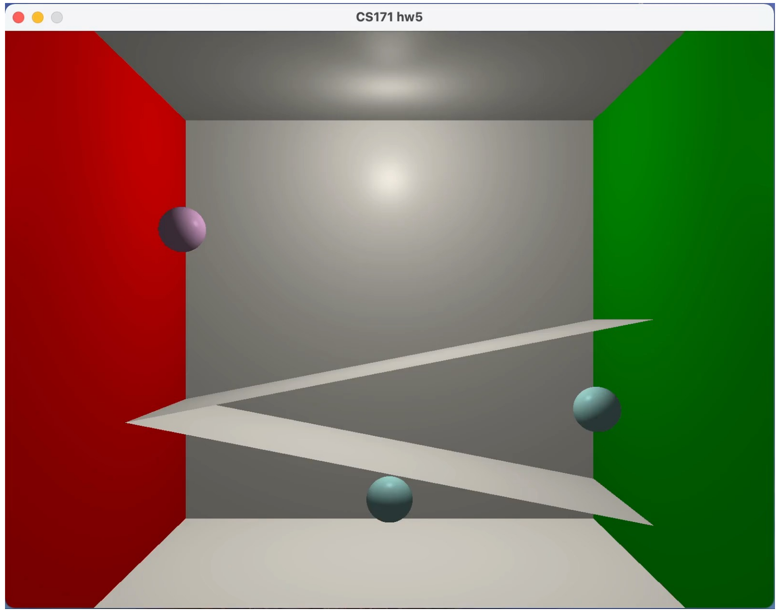
- Deadline: **22:00, Dec 25, 2021** 🎄
- Requirements
  - **[must]** Synchronize the simulation and the rendering in OpenGL to show the result in real time.
  - **[must]** Implement the collision detection between the parallelograms and the sphere.
  - **[must]** Implement the collision adjustment so that the inter-penetration is within the given tolerance (a small threshold).
  - **[must]** Implement the colliding contact handling between the the parallelogram and the sphere without consideration of rotation.
  - **[optional]** Simulate with multiple spheres simultaneously.
  - **[optional]** Simulate a cube in the scene with rotation taken into account.

Homework 5 Gallery

- After finishing the requisite parts

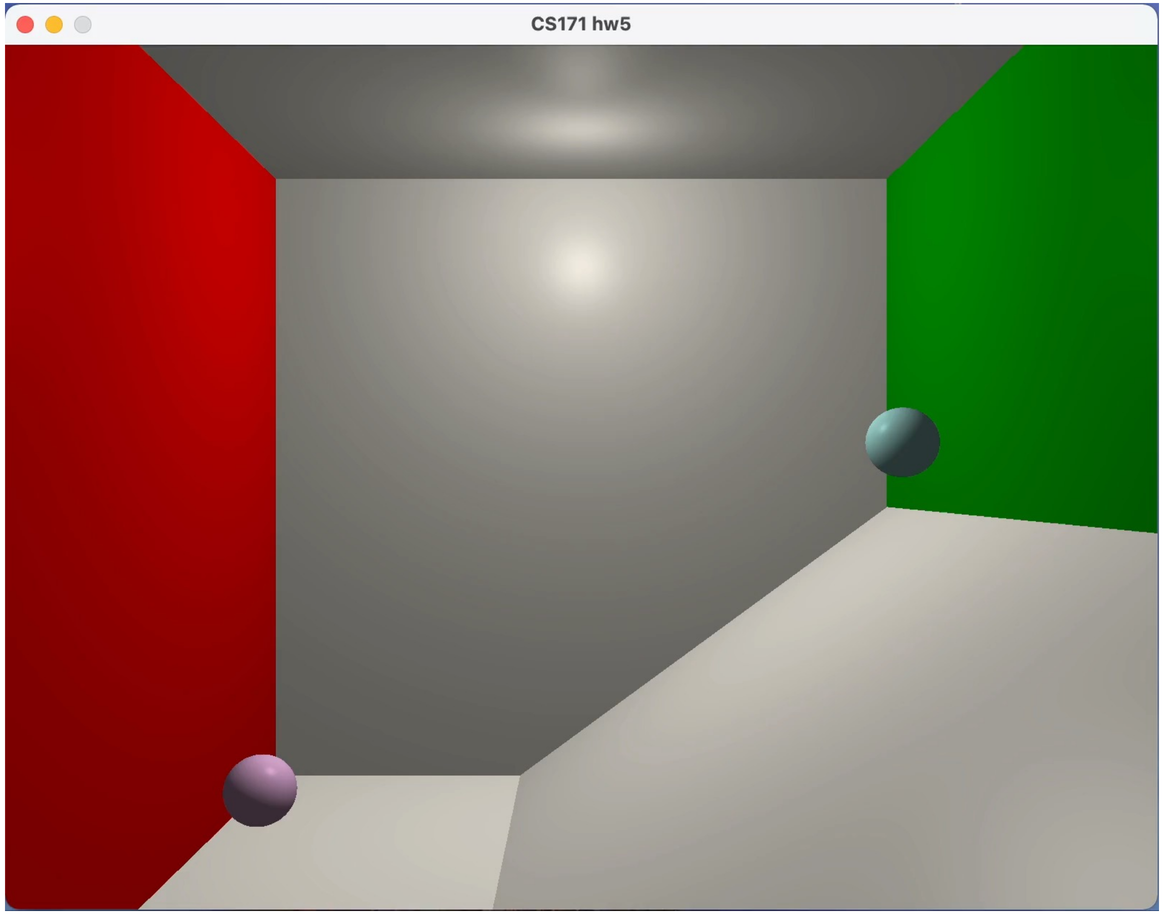


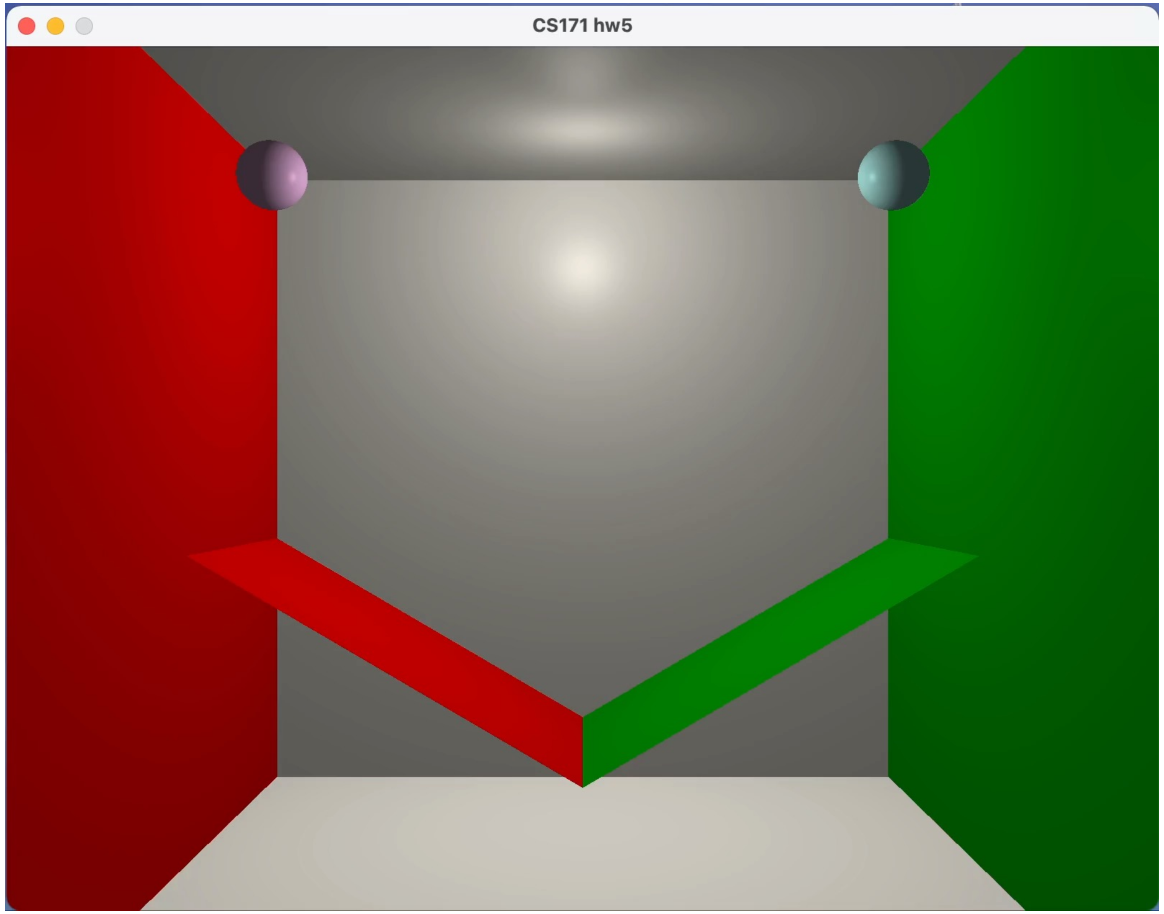




Homework 5 Gallery

- After implementing the sphere-sphere intersection

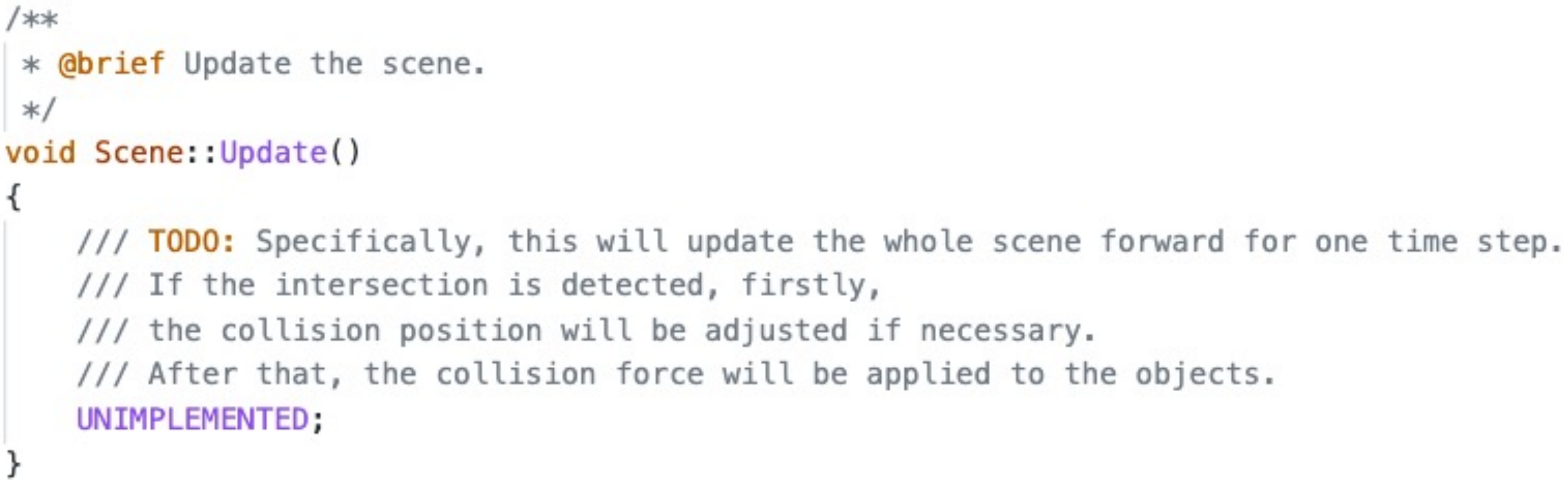




Code Structures

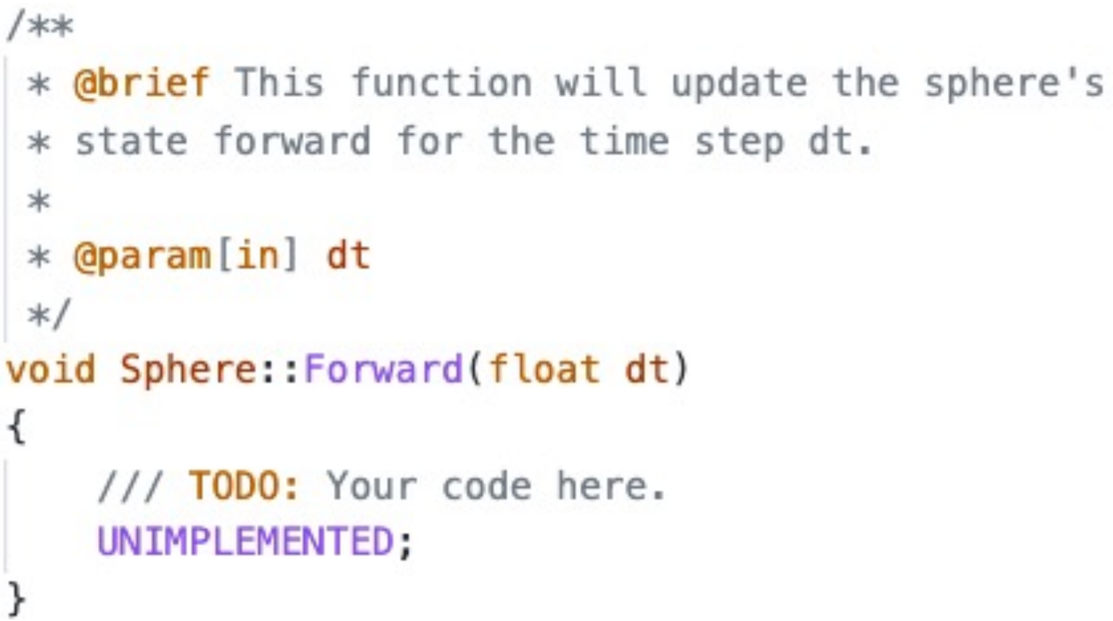
- **scene.h/cpp**: the whole scene to simulate
- **rigid_body.h/cpp**: some rigid bodies (wall, sphere) and their properties
- **collision.h/cpp**: collision handler
- **opengl_object.h/cpp**: opengl_object to facilitate the rendering with OpenGL
- **camera.h/cpp**: camera class
- **defines.h**: define some constants
- **shader.h**: shader class
- **utils.h**: define utility functions

Interfaces to Implement

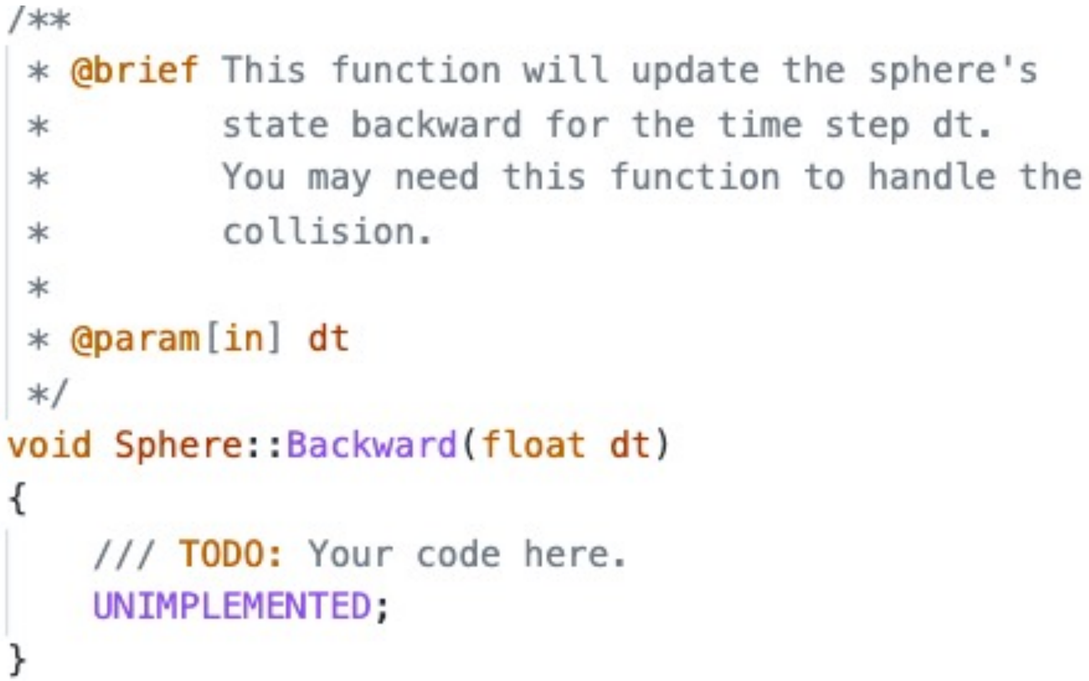


```
/**
 * @brief Update the scene.
 */
void Scene::Update()
{
    /// TODO: Specifically, this will update the whole scene forward for one time step.
    /// If the intersection is detected, firstly,
    /// the collision position will be adjusted if necessary.
    /// After that, the collision force will be applied to the objects.
    UNIMPLEMENTED;
}
```

Interfaces to Implement

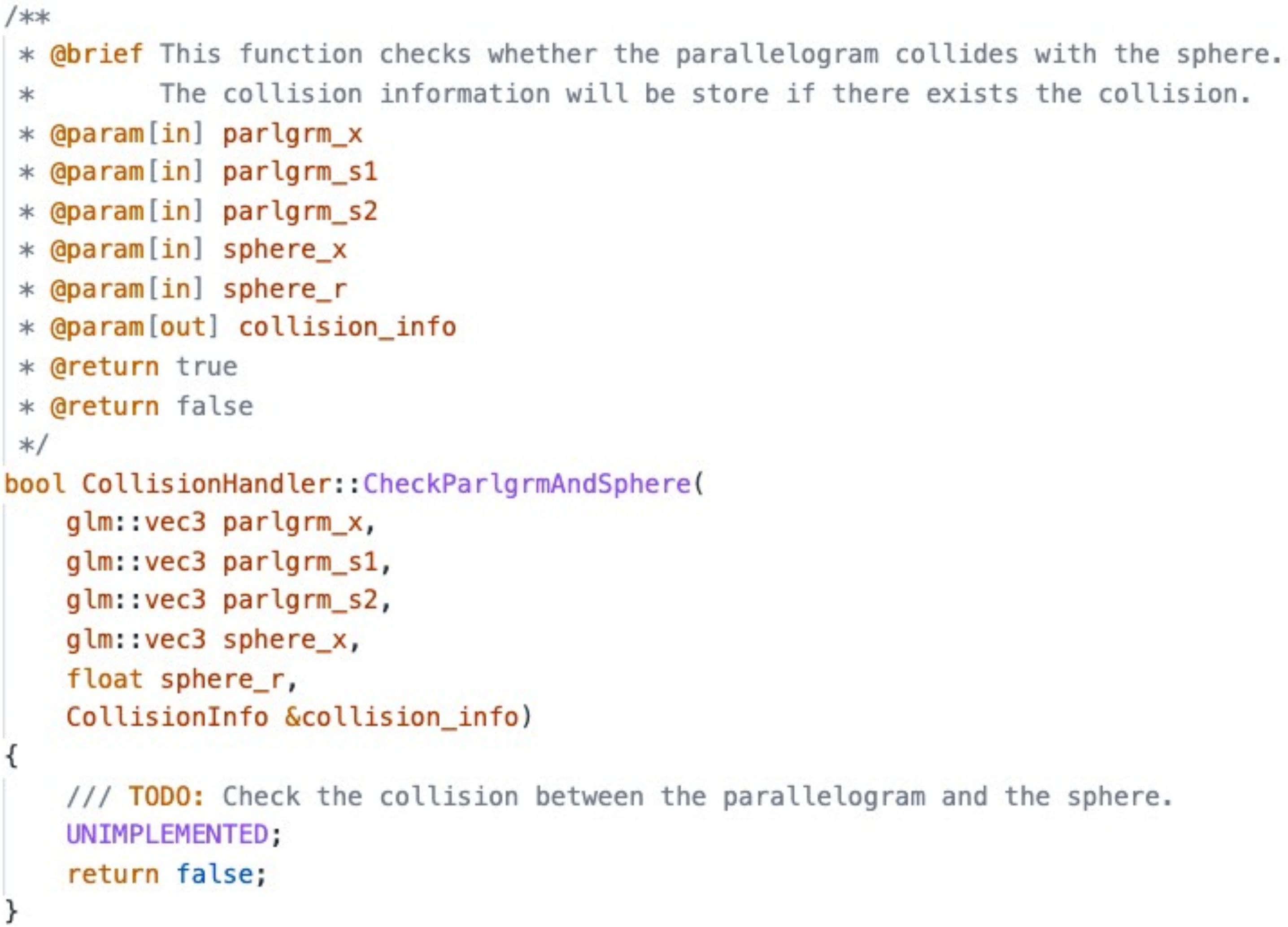


```
/**
 * @brief This function will update the sphere's
 * state forward for the time step dt.
 *
 * @param[in] dt
 */
void Sphere::Forward(float dt)
{
    /// TODO: Your code here.
    UNIMPLEMENTED;
}
```



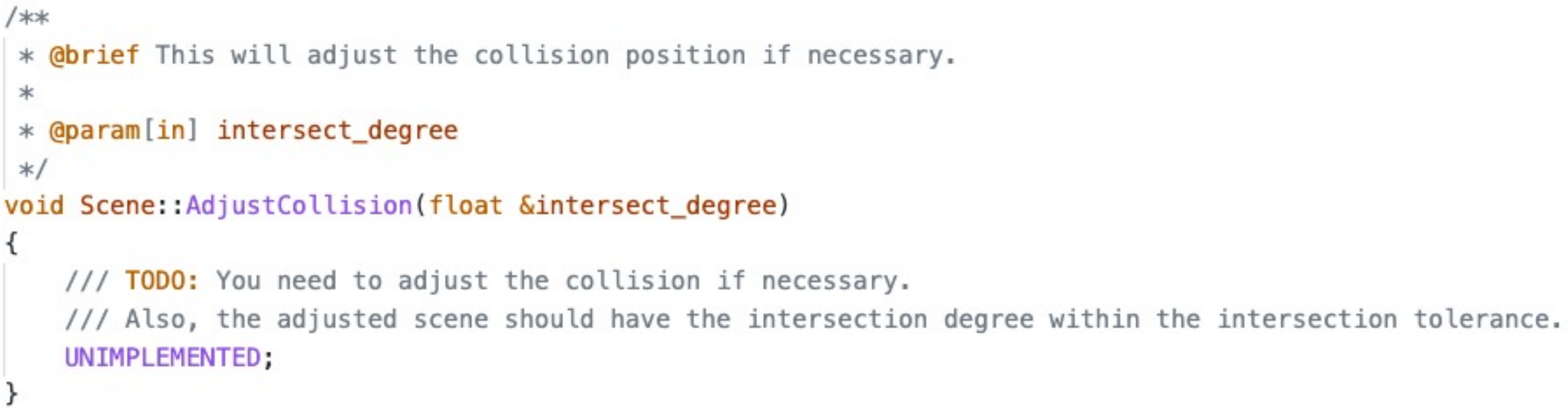
```
/**
 * @brief This function will update the sphere's
 *        state backward for the time step dt.
 *        You may need this function to handle the
 *        collision.
 *
 * @param[in] dt
 */
void Sphere::Backward(float dt)
{
    /// TODO: Your code here.
    UNIMPLEMENTED;
}
```

Interfaces to Implement



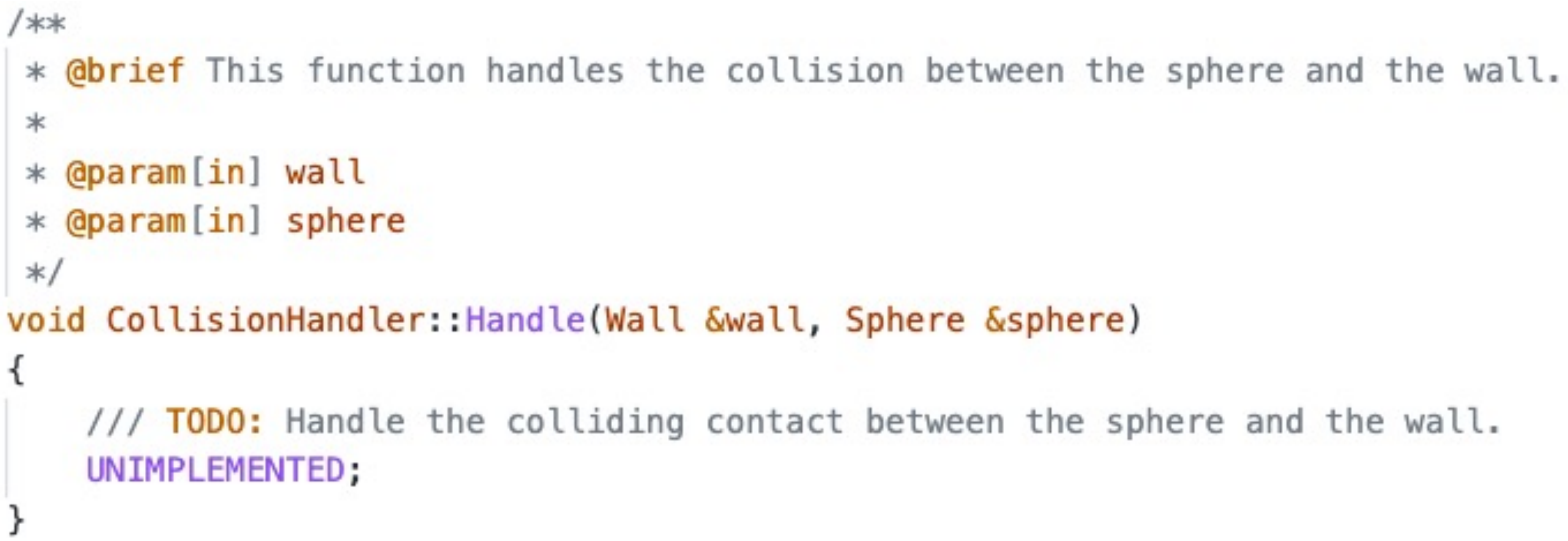
```
/**
 * @brief This function checks whether the parallelogram collides with the sphere.
 *        The collision information will be store if there exists the collision.
 * @param[in] parlgrm_x
 * @param[in] parlgrm_s1
 * @param[in] parlgrm_s2
 * @param[in] sphere_x
 * @param[in] sphere_r
 * @param[out] collision_info
 * @return true
 * @return false
 */
bool CollisionHandler::CheckParlgrmAndSphere(
    glm::vec3 parlgrm_x,
    glm::vec3 parlgrm_s1,
    glm::vec3 parlgrm_s2,
    glm::vec3 sphere_x,
    float sphere_r,
    CollisionInfo &collision_info)
{
    /// TODO: Check the collision between the parallelogram and the sphere.
    UNIMPLEMENTED;
    return false;
}
```

Interfaces to Implement



```
/**
 * @brief This will adjust the collision position if necessary.
 *
 * @param[in] intersect_degree
 */
void Scene::AdjustCollision(float &intersect_degree)
{
    /// TODO: You need to adjust the collision if necessary.
    /// Also, the adjusted scene should have the intersection degree within the intersection tolerance.
    UNIMPLEMENTED;
}
```

Interfaces to Implement



```
/**
 * @brief This function handles the collision between the sphere and the wall.
 *
 * @param[in] wall
 * @param[in] sphere
 */
void CollisionHandler::Handle(Wall &wall, Sphere &sphere)
{
    /// TODO: Handle the colliding contact between the sphere and the wall.
    UNIMPLEMENTED;
}
```

Interfaces to Implement

- You may need some more helper functions
- For example, you may need helper functions to facitilate
  - collision detection
  - Collision handling
- Feel free to add your own functions!

Overview of the Whole Workflow

- Initialize the whole scene
  - Initialize parameters
    - e.g. time step, velocity decays, tolerance threshold…
  - Initialize all rigid bodies inside
    - State variables: linear velocity/momentum
    - If considering the rotation
      - Inertia tensor (and its inverse) as well
      - Angular velocity/momentum
    - Initialize opengl object: VAO, VBO, EBO…

Overview of the Whole Workflow

- Update the scene in the main loop
  - Two aspects:
    - Update simulation state varisbles of each rigid body
    - Update the output of OpenGL based on the updated state variables
  - Scene::Update
    - Move each rigid object forward: RigidBody::Forward
    - But there might be collision after you push every rigid body forward…
    - So you need to deal with the possible collision.

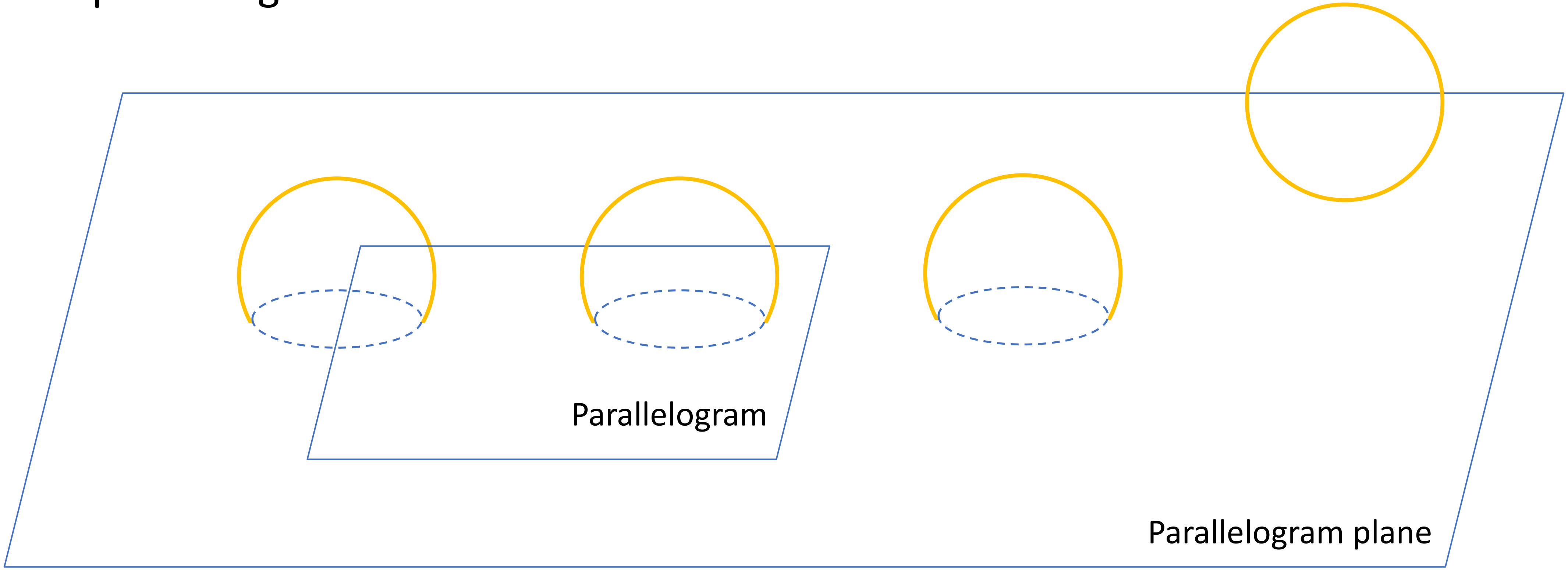
Overview of the Whole Workflow

- Update the scene in the main loop
    - Scene::Forward
    - Detect the collision between **every pair** of the objects
        - CollisionHandler::Check
            - CollisionHandler::CheckParlgrmAndSphere
    - If there is collision, you need to make sure the intersection degree is within the given tolerance: Scene::AdjustCollision
    - Scene::ApplyCollision
        - Handle the collision between **every pair** of the objects
            - CollisionHandler::Handle

Code

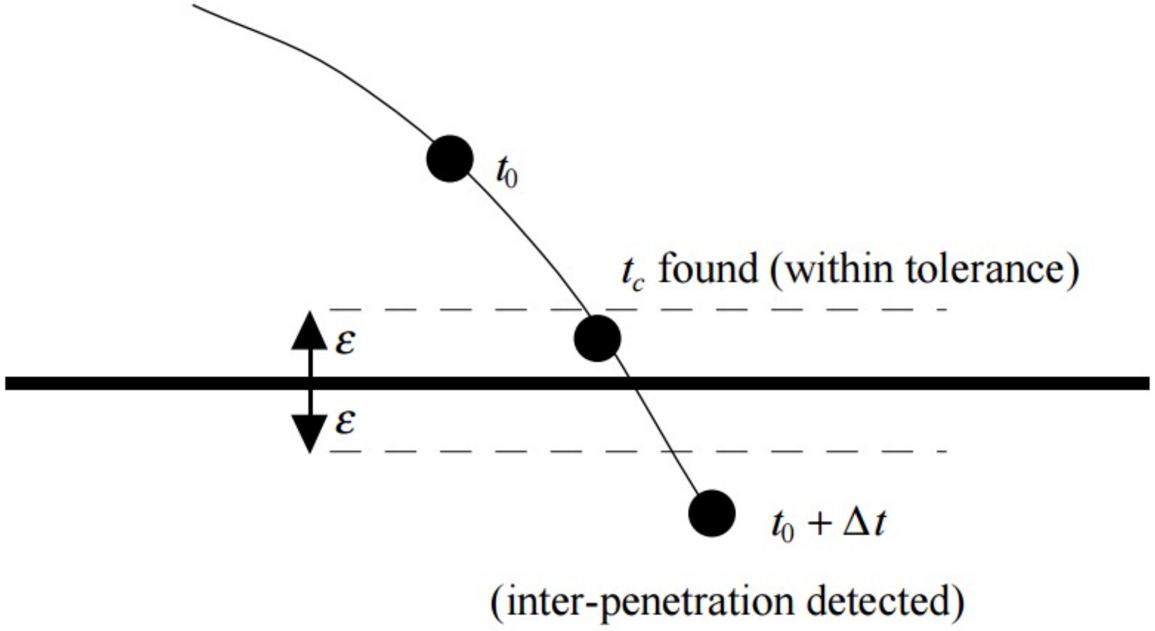
Collision Detection

- Between Sphere and parallelogram
  - Step1: Check whether the sphere intersects the parallelogram plane
  - Step2: If the sphere intersects the plane, check whether the sphere intersects parallelogram



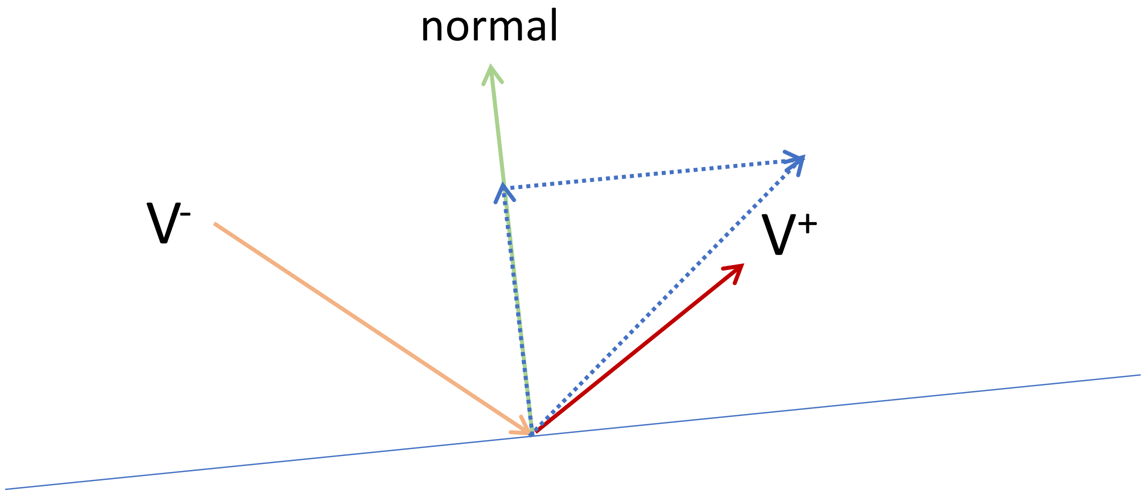
Collision Adjustment

- We want to search t_c between t_0 and $t_0 + \Delta t$
- Check $t_0 + \frac{1}{2}\Delta t$
  - if there is no collision, search between $t_0 + \frac{1}{2}\Delta t$ and $t_0 + \Delta t$
  - Else if there is collision but it is beyond the tolerance, search between t_0 and $t_0 + \frac{1}{2}\Delta t$
  - Otherwise, t_c is found
- Repeat
- You need to implement Backward



Collision Handling

- Here we do not consider rotation of spheres
- In our model, velocity will be decayed along both the normal and the tangent directions
- The impulse applied to the sphere: $(V^+ - V^-)M$



Collision Handling

- For simplicity, our wall is modeled as a parallelogram whose thickness is zero. If the sphere hits the wall from the lateral, it may be problematic because we don’t have a right definition of the nomral.
- If you want to make your code more robust, you may need some special treatments to handle this situation.
- To make it more generalized, the better way is to model it as a box with six sides.
- You can explore more on the optional task.
- ⛽ Good luck!

Thanks