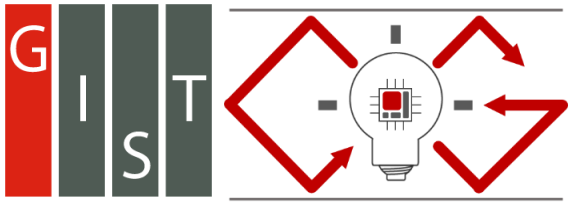
Programming Assignment 3

2026 Computer Graphics



Computer Graphics
Laboratory

2026/05/04

Seongil Kim

Contents

Notice

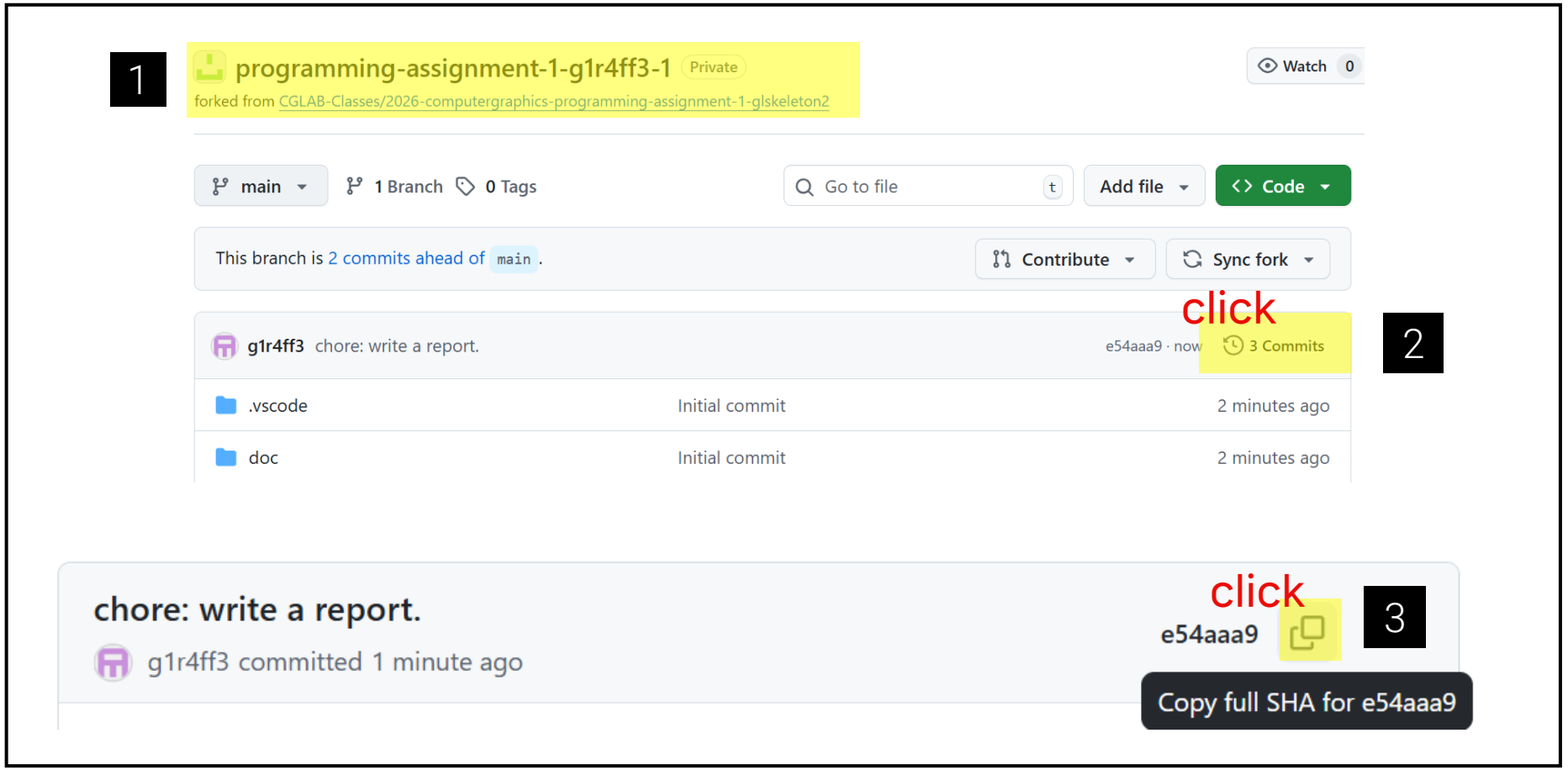
Submission

- Deadline: 23:59:59, Sunday, May 24th, 2026 (KST, +0900)
  - GitHub server clock

- To submit your assignment, two things must be done BEFORE deadline.
  - You must push your commit to your repository
  - You must comment the last commit id (SHA-1 hash) in github issue board
  - You must match username of pushed commits with GitHub username

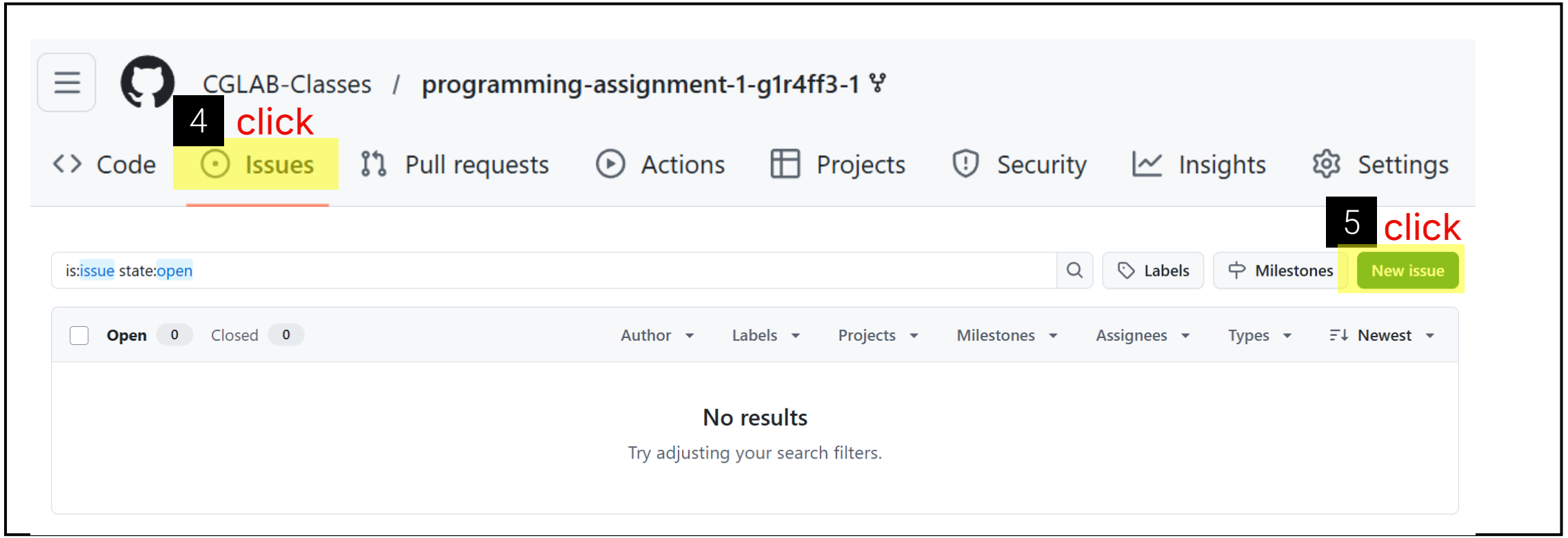
- The last commit in the issue board BEFORE deadline will be considered as submitted assignment.
  - Local timestamp in your commit will be ignored. GitHub server timestamp used instead

How to comment a commit id



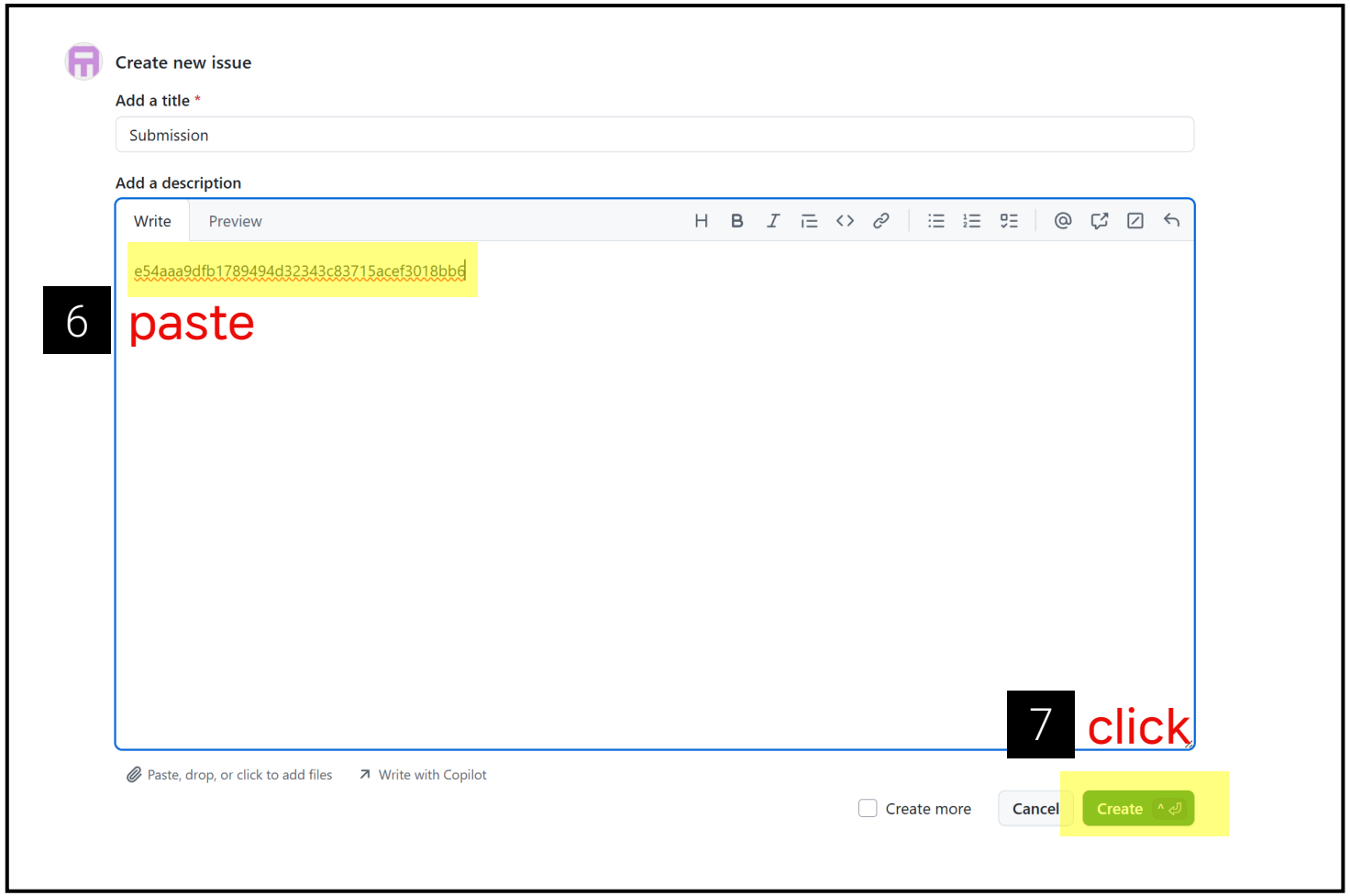
1. Go to your assignment repository
2. Click "Commits"
3. Click copy button of your last commit

How to comment a commit id



4. Go to issues tab
5. Click "New Issue"

How to comment a commit id



6. Paste your latest commit id (ctrl-v)
7. Click "Create"

Policies

- In following cases, your grade for this PA will be "0"
  - Late submission
    - late push after deadline or late last commit id comment on issue board
  - Build/execution failure
  - Making public of your assignment repository
  - Mismatch of username in a submitted local commit with the submitter's GitHub username
  - API use of a library that gives the solution directly, instead of working your assignment on your own

- Your final grade will be "F"
  - Copy (We will run a copy detector)

Assignment Overview

Task 1. Whitted Ray Tracer

- You will implement Whitted ray tracer that can generate lighting effects like reflections or refractions
  - Refer to the "Whitted Ray Tracing" lecture note in the course page or the original paper [1]

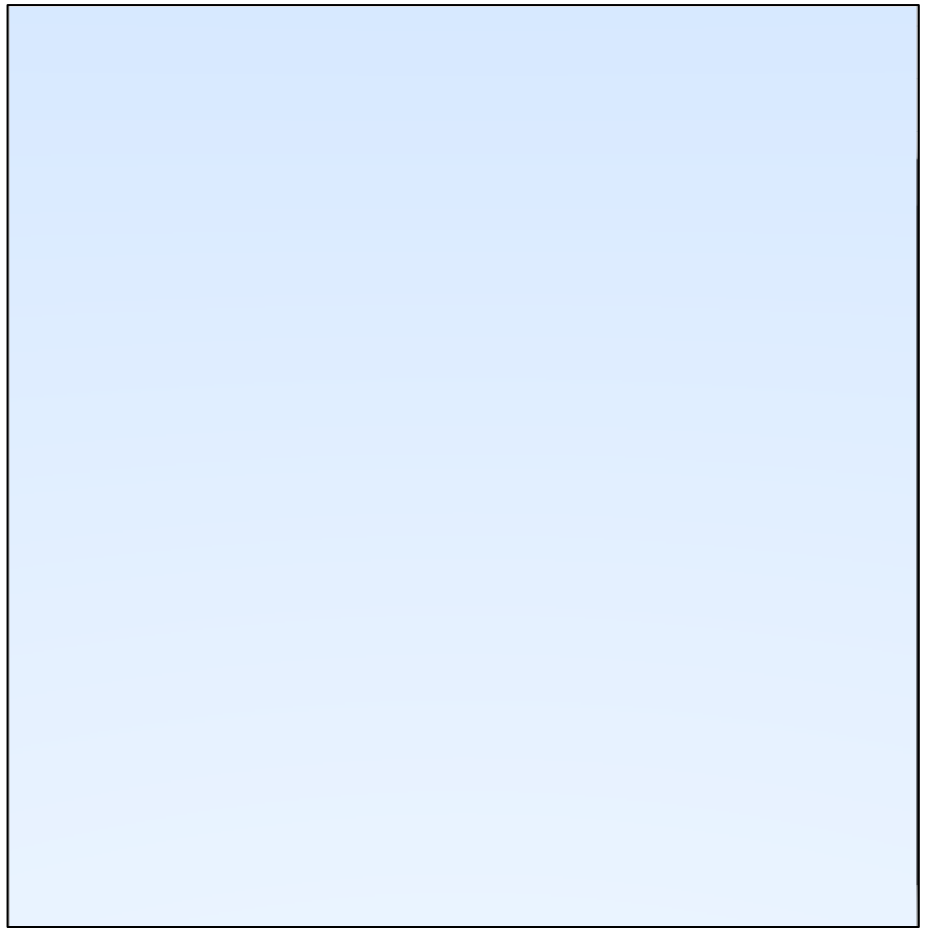
- Implementation constraints
  1. Generate two secondary rays at surface, one for reflection and the other for refraction
  2. Use the provided material classes for shading: WhittedMaterial/CheckerMaterial
  3. Determine how refractive rays interact at the surface
     - Total internal reflection [2]: the refraction case falls back to a reflection case
     - Fresnel equation (or Schlick approximation): how much the light energy should be distributed into reflection and refraction

- Note
  - The Whitted ray tracer is not based on physics; it is a hybrid method that balances local and global illumination models, i.e. deterministic ray tracer
  - From this assignment, a rendering result is a single image not a sequence of images (i.e. video)

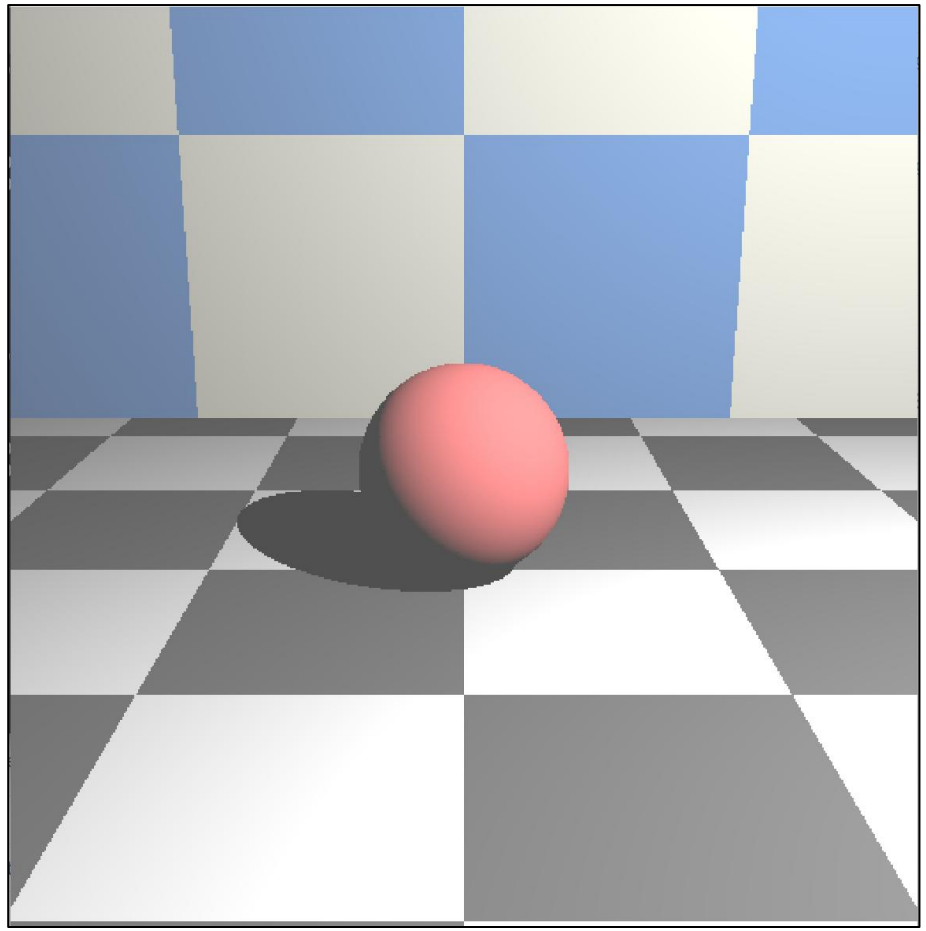
[1] T. Whitted. An improved illumination model for shading display. Communications of the ACM, 23(6):343–349, 1980
[2] "Ray Tracing in One Weekend." raytracing.github.io/books/RayTracingInOneWeekend.html (accessed May. 1, 2026)

Initial Appearance

- Skeleton code is only able to draw sky-gradient background for a sample scene (scenes/sample.xml)



Task 1 finished



Scene description with xml

- From this assignment, you can describe a scene using xml as follows:
  - Top level tag: this must be a scene
  - Fields in each tag: you can find how to specify the parameters in their codes
  - For more information, refer to README.md in the repository

scenes > sample.xml

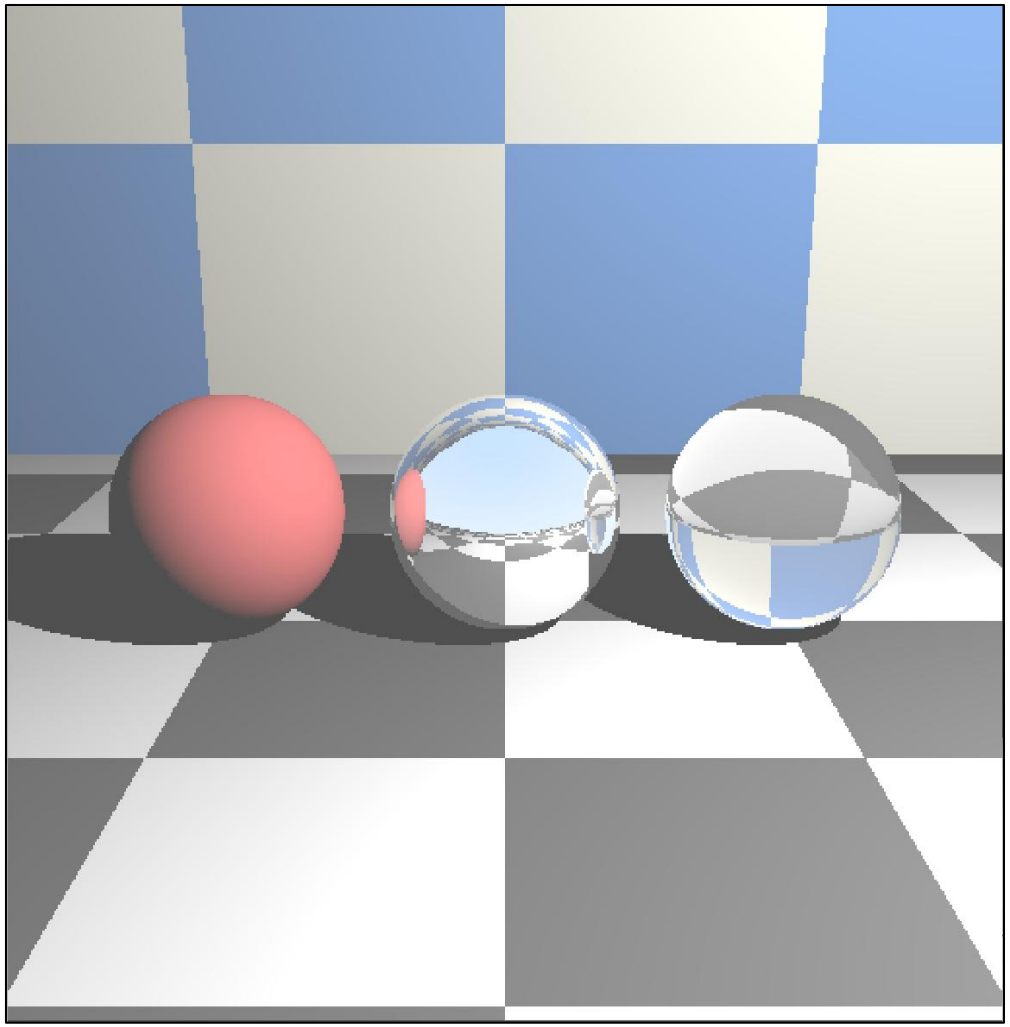
```xml
<!-- Sample scene: one diffuse sphere on the same checker stage used by
     scenes/sphere/front.xml and scenes/ajax/front.xml. Renders as the sky
     gradient until Li(), Sphere::rayIntersect, and the acceleration
     structure are implemented. -->
<scene>
    <integrator type="whitted_integrator">
        <integer name="maxDepth" value="8"/>
    </integrator>

    <camera type="perspective">
        <integer name="width" value="512"/>
        <integer name="height" value="512"/>
        <float name="fov" value="45"/>
        <transform name="toWorld">
            <lookat origin="0, 2, 5" target="0, 0.5, 0" up="0, 1, 0"/>
        </transform>
    </camera>

    <!-- Checker floor -->
    <mesh type="obj">
        <string name="filename" value="../meshes/plane.obj"/>
        <material type="checker">
            <color name="color1" value="0.9, 0.9, 0.9"/>
            <color name="color2" value="0.15, 0.15, 0.15"/>
            <float name="scale" value="0.8"/>
        </material>
    </mesh>
```

src/perspective.cpp

```xml
    <!-- Checker back wall -->
    <mesh type="obj">
        <string name="filename" value="../meshes/plane.obj"/>
        <transform name="toWorld">
            <rotate angle="90" axis="1, 0, 0"/>
            <translate value="0, 3, -3"/>
        </transform>
        <material type="checker">
            <color name="color1" value="0.2, 0.4, 0.8"/>
            <color name="color2" value="0.95, 0.95, 0.85"/>
            <float name="scale" value="0.5"/>
        </material>
    </mesh>

    <!-- Diffuse (red) sphere — center -->
    <primitive type="sphere">
        <point name="center" value="0, 0.5, 0"/>
        <float name="radius" value="0.5"/>
        <material type="whitted">
            <color name="Kd" value="0.85, 0.2, 0.2"/>
        </material>
    </primitive>

    <light type="point">
        <point name="position" value="3, 4, 3"/>
        <color name="intensity" value="40, 40, 40"/>
    </light>

    <color name="ambient" value="0.08, 0.08, 0.08"/>
</scene>
```

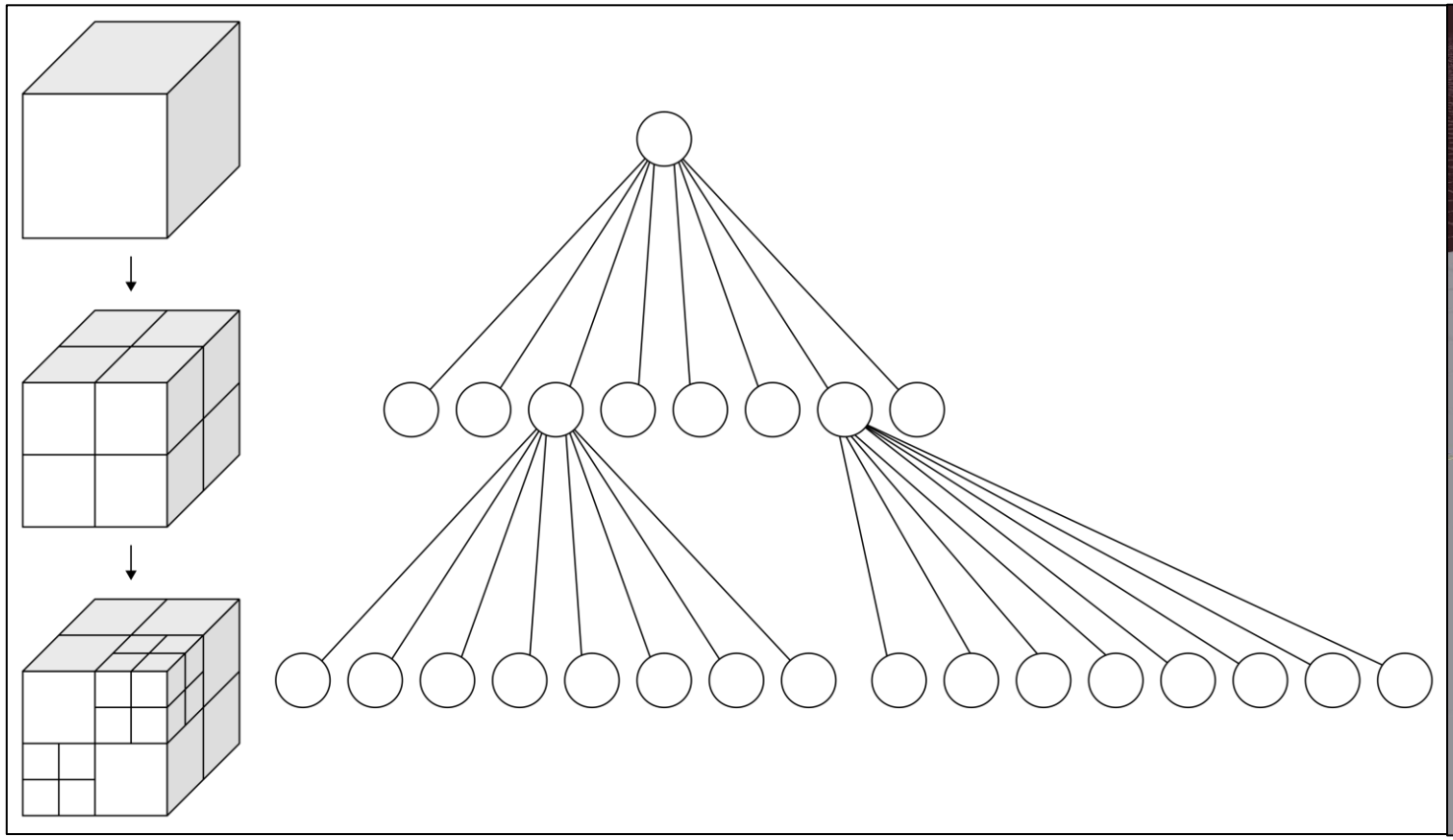
Test Scene: Sphere

- Spheres are present in a row with different material parameters (scenes/sphere/front.xml)
  - Checker material is provided; Check your skeleton repository



Task 2. Octree

- Hierarchical spatial data structure that
  - divides space of whole scene or single geometry by octants recursively
  - enables fast searching an intersection point of a dense geometry in O(log N) time



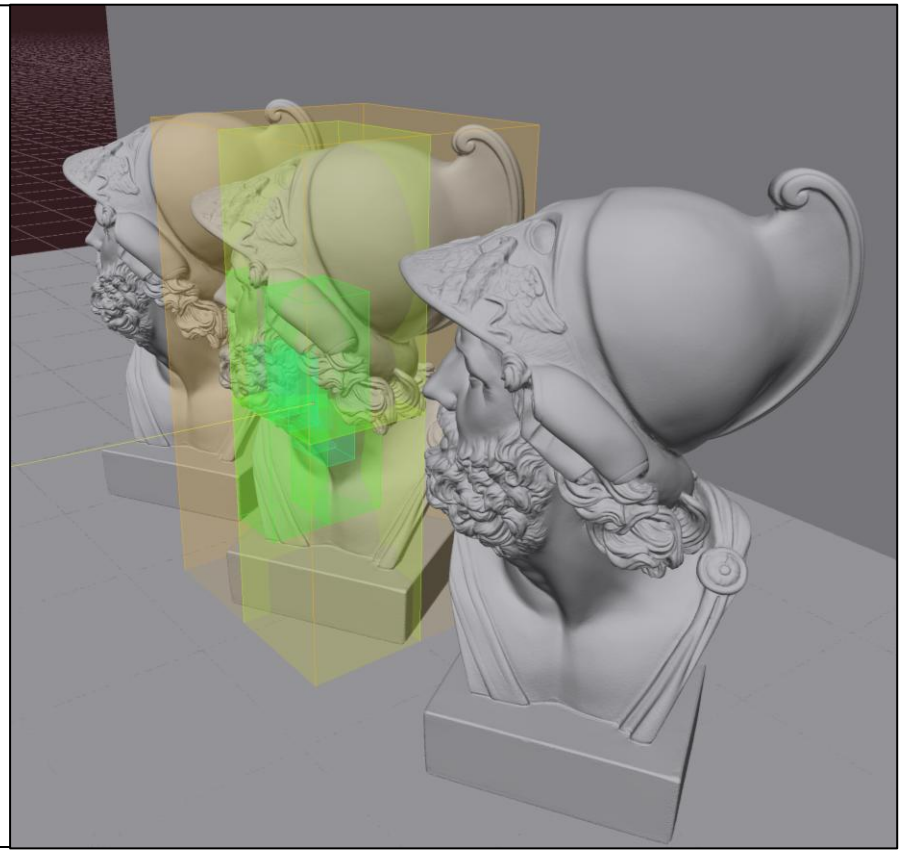


Image source: https://en.wikipedia.org/wiki/Octree

Octree Construction

- Algorithm
  1. Start with a root bounding box at scene level
  2. Split the current box by 8 octants
  3. For primitives, test overlaps with their children and classify them into their appropriate nodes
  4. Recursively do the items 2 & 3 for children until a base condition is met
     - For example, you can stop splitting a box with fewer than 10 triangles

1
```
Node *build(bounding box, triangles) {
    if (no triangles)
        return nullptr;

    if (only few triangles)
        return new leaf node with triangles;
```
"few triangles" means less than 10 triangles

2
```
triangle_list list[8];
```

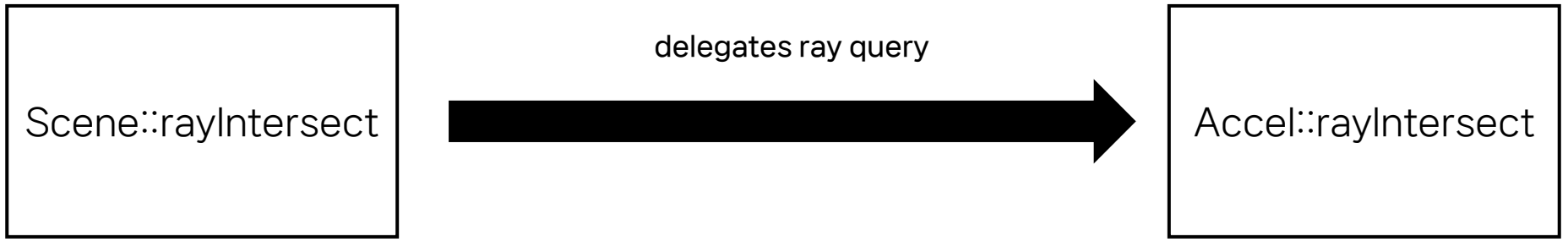
3
```
for (every triangle) {
    for (int i = 0; i < 8; ++i) {
        if (triangle overlaps sub-node i)
            add to list[i];
    }
}
```

4
```
Node *node = new Node();
for (int i = 0; i < 8; ++i)
    node.child[i] = build(bounding box of sub-node i, list[i]);
return node;
}
```

Figure 1. pseudocode for building an octree when there are only meshes

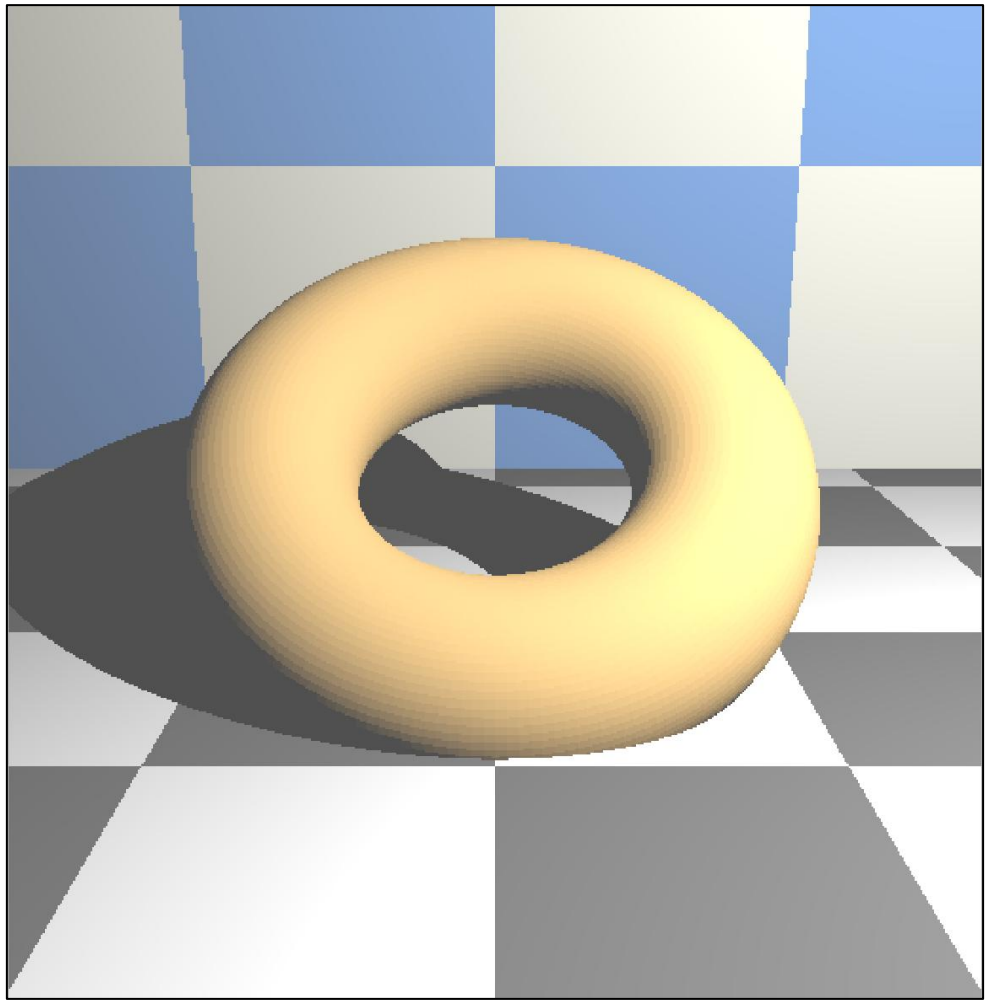
Ray traversal

- Once you have built the octree, you need to know how to trace it
  - Goal: figure out how to implement the ray traversal
- Algorithm
  1. Test the ray against the current node's bounding box (AABB)
  2. If the ray hit a leaf node, intersect the ray with all contained primitives
  3. Otherwise, recurse into children sorted by ray hit distance



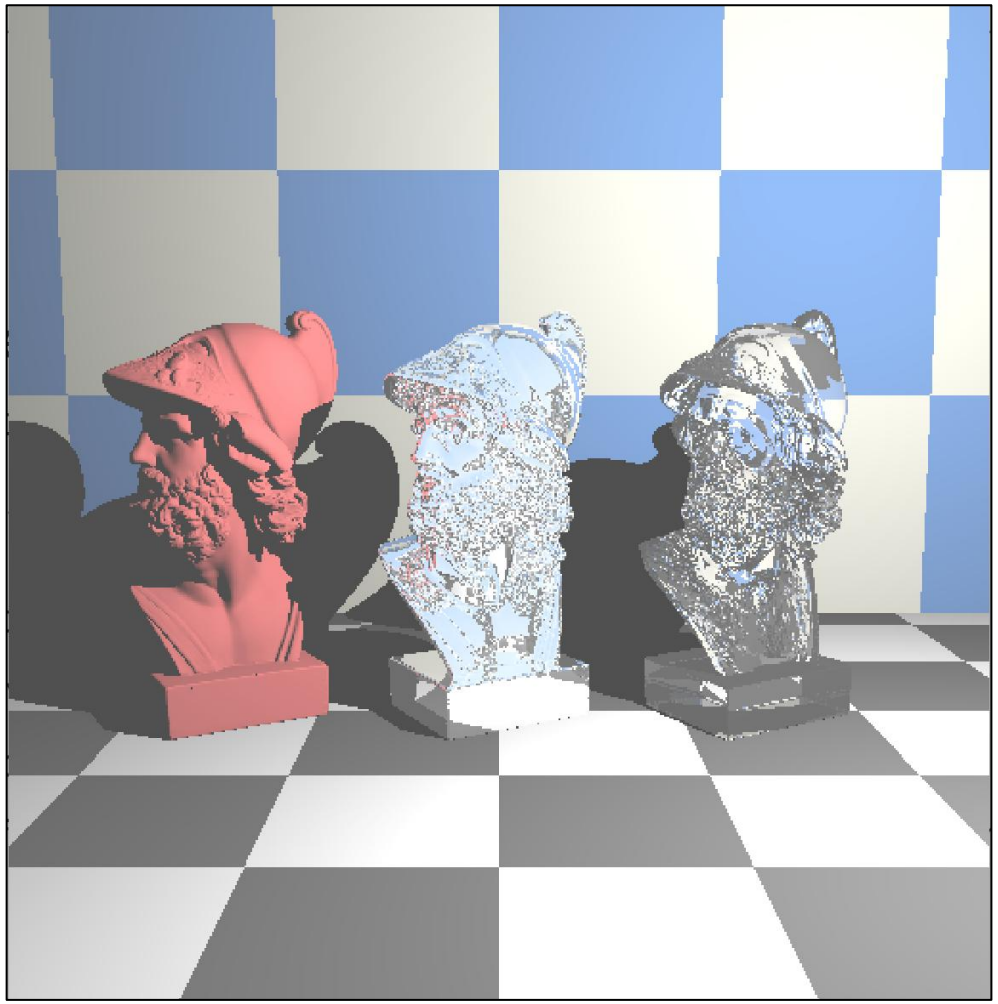
Development Scene: Torus

- Torus mesh with about 30K triangles is displayed in front of the camera (scenes/torus/front.xml)
  - Checker material is provided; Check your skeleton repository

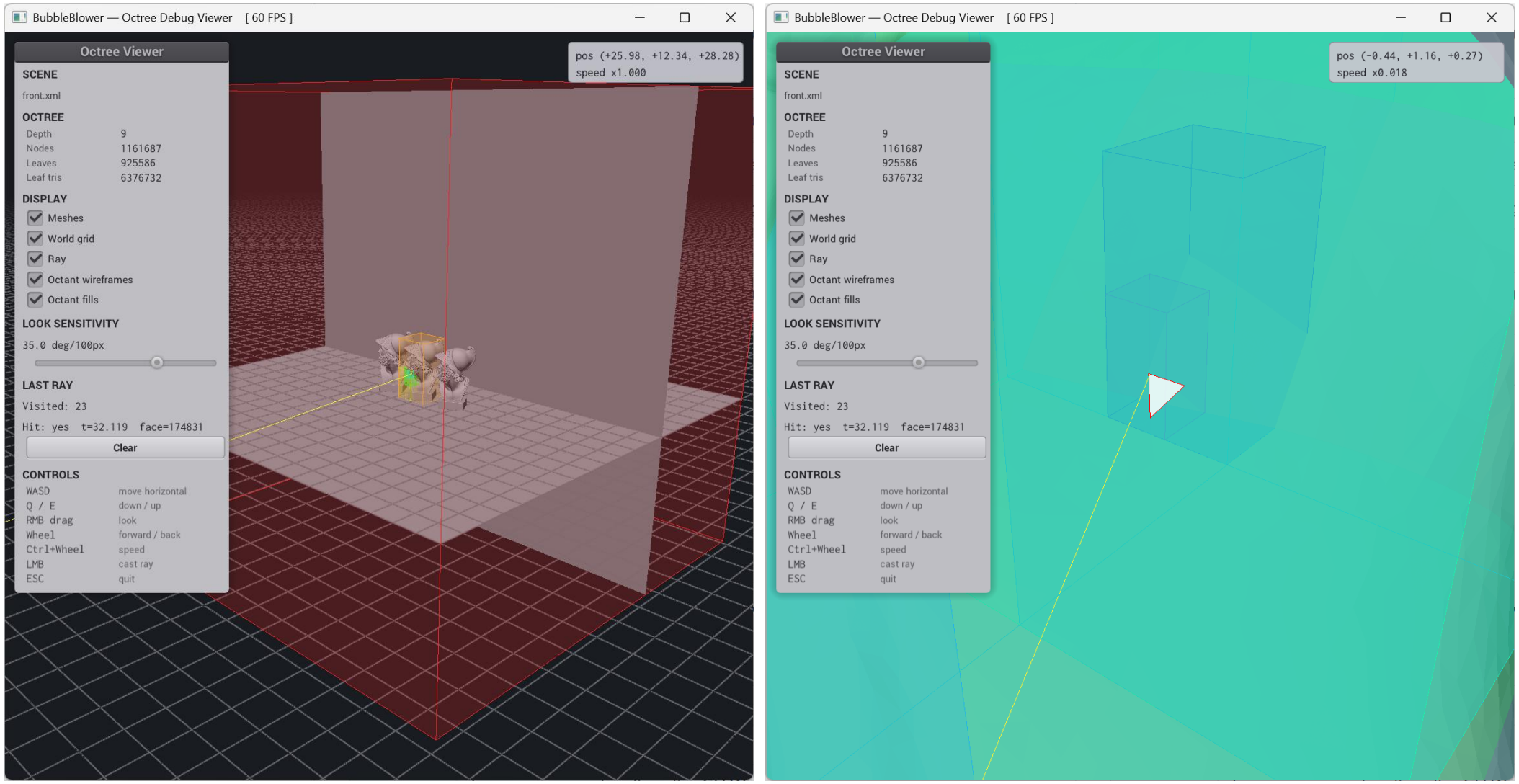


Test Scene: Ajax bust

- Ajax busts are present in a row with different material parameters (scenes/ajax/front.xml)
  - Checker material is provided; Check your skeleton repository



Demo: Octree Visualization



Sample views of the Ajax bust test scene

Task List

1. Implement Whitted ray tracer and verify it by the sphere test scene [16 Points]
   - Use the sphere test scene to prepare result images for the report [1 Points]
   - Render the diffuse sphere correctly [5 Points]
   - Render the mirror sphere correctly [5 Points]
   - Render the glass sphere correctly [5 Points]
2. Implement the octree and verify it by the ajax bust test scene [13 Points]
   - Use the ajax bust test scene to prepare result images for the report [1 Points]
   - Render the diffuse ajax correctly [3 Points]
   - Render the mirror ajax correctly [3 Points]
   - Render the glass ajax correctly [3 Points]
   - Speed-up more than 10% relative to the simple Whitted ray tracer is achieved when using the octree [3 Points]
3. Write a report [1 Points]
   - Describe your submission in detail; anything is fine, but it should be meaningful and persuasive [1 Points]
     - E.g., implementation details, pseudocode for your algorithms, analyses about memory efficiency, computation speed or performance-related metrics, or etc.
     - Attach rendering images to back up your arguments
     - Write your name, student id, GitHub username

Accept your PA3!
https://classroom.github.com/a/T1h0bJGs

Q & A

- Email
  - Seongil Kim: sngillkym@gm.gist.ac.kr
  - Hyunjin Jung: hjjung0810@gm.gist.ac.kr
- Office: 104 Dasan Bldg