CS350 Project 2

Synopsis

In preparation for spatial data structures, add several features to the framework:

- Debug drawing for types: Segment, Triangle, Box, and Sphere.
- Collision detection
- Triangle selection
- Establish a timing baseline.

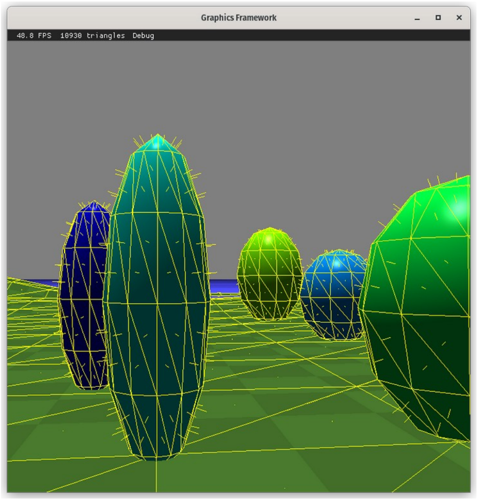


Figure 1: Triangle and segment draws.

Instructions

This project has five steps.

1. Preparation:

- In **student_code.h**, ensure **MAIN** and **GOAL** are set correctly for this project:

  #define MAIN SCAN_LINE
  static int GOAL = 10*thousand; // or smaller.

- Find procedures **ObjectTriangles** and **EndOfTriangles** in **student_code.cpp** and follow the instructions there to create a simple list of triangles. (Suggesting: **std::vector<Triangle*>**.)

- Since you will eventually need a bounding box for each triangle, this is a good time to create a list of Box* also.

2. Debug Drawing:

In **student_code,h** and **student_code.cpp**, find eight instances of the comment

// @@ Example debug for Segments. ... |

and use the **Segment** drawing code found there as a road map to implementing similar functionality for **Triangle**, **Box**, and **Sphere**.

Your goal is to draw images similar to figures 1 and 2, though you may choose to draw far fewer instances if your CPU/GPU are stressed by this full example.

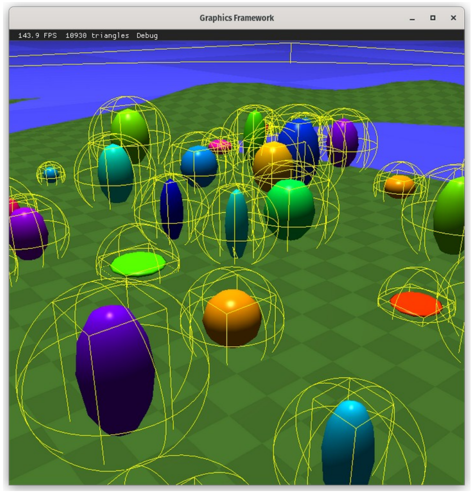


Figure 2: Box and Sphere draws.

3. Collision Detection:

In student_code.cpp, find and implement

bool StudentCode::IsStepLegal(vec3 eye, vec3& direction, float step) {...}

It is expected to compare the **Segment(eye, eye+step*direction)** to every triangle on the above list and return **false** (to prevent the proposed step) if the segment intersects any triangle. Otherwise a **true** is returned to allow the system to take the step.

4. Triangle Selection:

In **student_code.cpp**, find procedure

void DrawDebug(unsigned int programId) {…}

This is called after the main drawing with the expectation that you can do all your debug drawing here.

For this portion of the project, find the first triangle visible along the center-of-screen view ray and debug draw it as shown in figure 3. The code contains the following code which compute the view ray, and a class lecture will describe how these calculations achieve that.

```
vec3 eye = Hdiv(ViewInverse*vec4(0,0,0,1));
vec3   T = Hdiv(ViewInverse*ProjInverse*vec4(0,0,-1,1));
Ray ray(eye, T-eye);
```

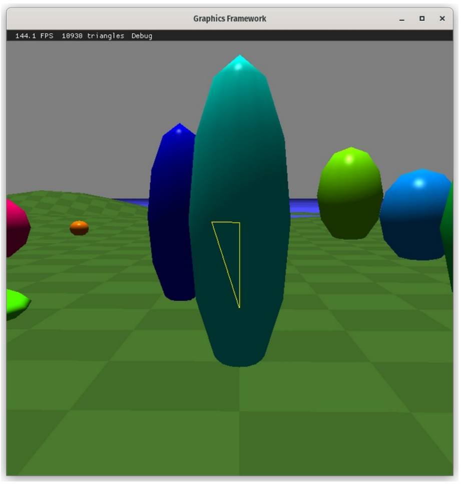


Figure 3: The single triangle visible at the center-of-screen.

5. Timing baseline:

At this point, each drawn frame requires two passes thorough the triangle list, one for collision detection and one for triangle selection. The goal here is to establish how long the triangle list can be before the framework suffers a frame-rate drop. When spatial data structures are introduced in later projects, we'll again increase the size of the triangle lists with the expectation that we will be able to handle far, FAR larger lists.

In **student_code.h**, the number of triangles is set (approximately) with this line:

static int GOAL = 10*thousand;

Increase that number in steps to determine how many triangles your system can handle without a frame rate slow down. There is no need for great accuracy here. The closest multiple of 10,000 triangles is sufficient.

6. Reporting FPS

In student_code.cpp, at the top of **StudentCode::DrawGui** do this:

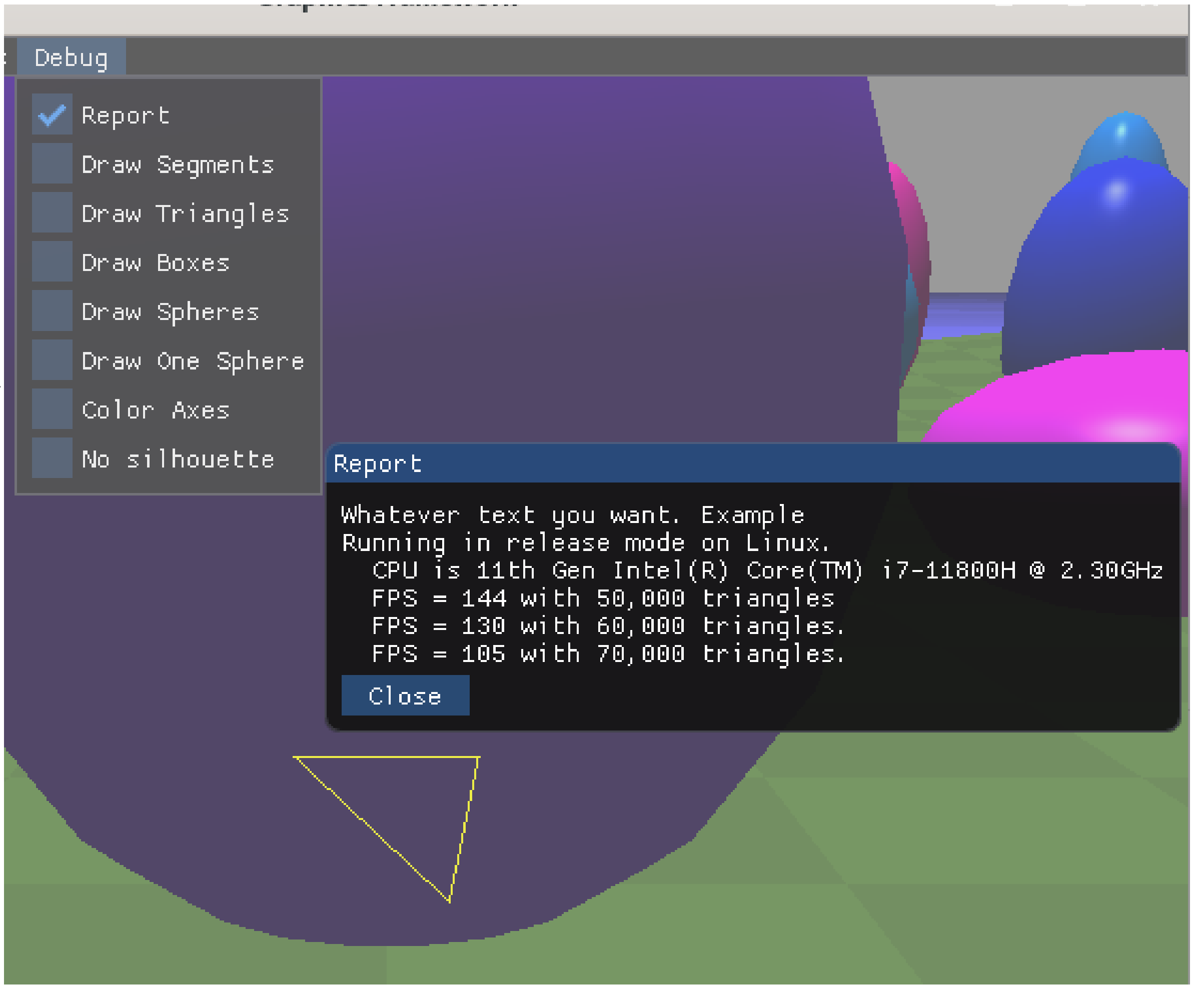
```
void StudentCode::DrawGui()
{
    static bool popup = false;
    if (ImGui::Checkbox("Report", &popup))  ImGui::OpenPopup("Report");

    if (ImGui::BeginPopupModal("Report", nullptr, ImGuiWindowFlags_AlwaysAutoResize)) {
        ImGui::Text("Whatever text you want. For example\n\
Running in release mode on Linux.\n\
  CPU is 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz\n\
  FPS = 144 with 50,000 triangles\n\
  FPS = 130 with 60,000 triangles.\n\
  FPS = 105 with 70,000 triangles.");
        if (ImGui::Button("Close", ImVec2(60, 0))) ImGui::CloseCurrentPopup();
        ImGui::EndPopup(); }

...
}
```

You can get the CPU information by running **msinfo32** in a command prompt on Windows, or **lscpu** on Linux.

If the **ImGUI** code above does not work for you, then create a simple text file named **report.txt** and include it in the submitted ZIP file.



```
Whatever text you want. Example
Running in release mode on Linux.
  CPU is 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz
  FPS = 144 with 50,000 triangles
  FPS = 130 with 60,000 triangles.
  FPS = 105 with 70,000 triangles.
```

What to submit

Submit to Moodle a single ZIP file containing any source code files you have changed, and nothing else.

- At least: **student_code.h** and **student_code.cpp**.
- At most: all source code (*.cpp and *.h and perhaps *.frag and *.vert)

Do not include any of the following:

- Build artifacts (*.o, *.pdb, *.sdf, etc...) ,
- Executable files (*.exe),
- Debug, Release, or .vs folders.