1. Introduction

I've made a couple of improvements to the project to provide a unified way to compile the project on Windows and Linux. I've included precompiled binaries for Windows and Linux.

a. *Compilation on Windows*

- Install Visual Studio from `https://visualstudio.microsoft.com/downloads/`
- Install VCPKG from `https://vcpkg.io/en/getting-started.html`
- Add VCPKG folder to PATH:
- Option 1: Command Line tools

```
REM Assuming that VCPKG cloned and bootstrapped in c:\src\vcpkg
REM setx for the global environment, set for the local
setx PATH c:\src\vcpkg;%PATH%
set  PATH c:\src\vcpkg;%PATH%
```

- Option 2: PowerShell

```
# Assuming that VCPKG cloned and bootstrapped in c:\src\vcpkg
[Environment]::SetEnvironmentVariable("PATH", "c:\src\vcpkg;${PATH}", "Machine")
    Set-Item -Path Env:PATH -Value "c:\src\vcpkg;${PATH}"
```

- Option 3: Manually in **System Properties –> Environment Variables**
- Navigate to the folder with the project
- Run *build.bat*
- Open *build*
  EDGSG.sln in Visual Studio to work with the source code

b. *Compilation on Linux*

- Install VCPKG from `https://vcpkg.io/en/getting-started.html`
- Add VCPKG folder to System PATH
- Option 1: Temporary local environment

```
# Assuming that VCPKG cloned and bootstrapped in ~/vcpkg
export PATH="~/vcpkg;${PATH}"
```

- Option 2: Local environment and Bash profile

```
# Assuming that VCPKG cloned and bootstrapped in c:\src\vcpkg
export PATH="~/vcpkg;${PATH}"
echo 'export PATH="~/vcpkg;${PATH}"' >> ~/.bashrc
```

- Navigate to the folder with the project
- Run build.sh

Important Note: the VCPKG requests installation of additional packages

2. Project organization

The two files with source code were added: **Scenes.h** and **Scenes.cpp** to allow granular management for scenes. In the Scene class defined static functions to create specific scene setups:

```
class Scenes
{
public:
    // Practice 0:
    static void p0(SceneContent& sc);
    static void p0a(SceneContent& sc);

    // Practice 1:
    ...
    ...
    ...
    // Practice 2:
    static void p2a(SceneContent& sc, int numPointClouds, int pointsPerCloud, float scaleFactor);
    static void p2b(SceneContent& sc);
    static void p2c(SceneContent& sc);

    // Practice 3:
    static void p3(SceneContent& sc);

    // Practice 4:
    static void p4a(SceneContent& sc, bool drawTriangles);
    static void p4b(SceneContent& sc, bool randomOnSphereSurface);
    static void p4c(SceneContent& sc);
};
```

These methods are used in SceneContent:

```
void AlgGeom::SceneContent::buildScenario()
{
    constexpr int      numPointClouds = 1;
    constexpr int      pointsPerCloud = 50;
    constexpr float    scaleFactor    = 1.0f;
    std::vector<Point> randomPointsFromCloud;
    std::vector<Point> extremumPointInCloud;

    // Practice 1:
    ...
    ...
    ...
    // Practice 2:
    // Scenes::p2a(*this, numPointClouds, pointsPerCloud, scaleFactor);
    // Scenes::p2b(*this);
    // Scenes::p2c(*this);

    // Practice 3:
    // Scenes::p3(*this);

    // Practice 4:
    // Scenes::p4a(*this, /*draw triangles*/ true);           // 2D Hull
    // Scenes::p4b(*this, /*random on sphere surface*/ true); // 3D Hull
    Scenes::p4c(*this);  // 3D Hull from Trianles
}
```

3. Convex Hull 2D

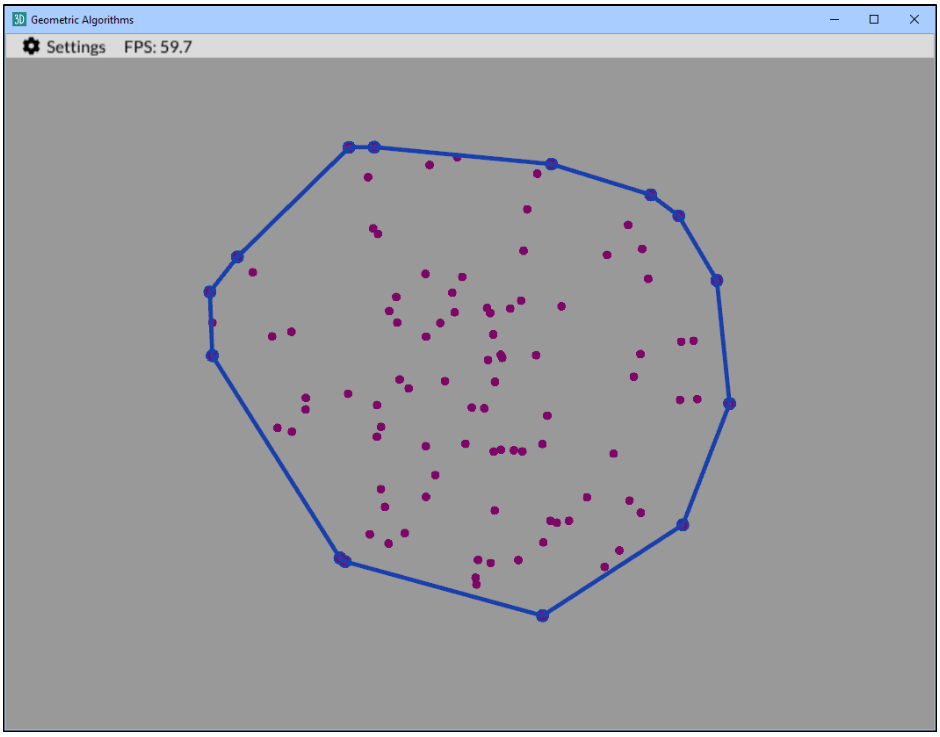

Figure 1: Point Cloud 2D to Convex Hull 2D (only outline)

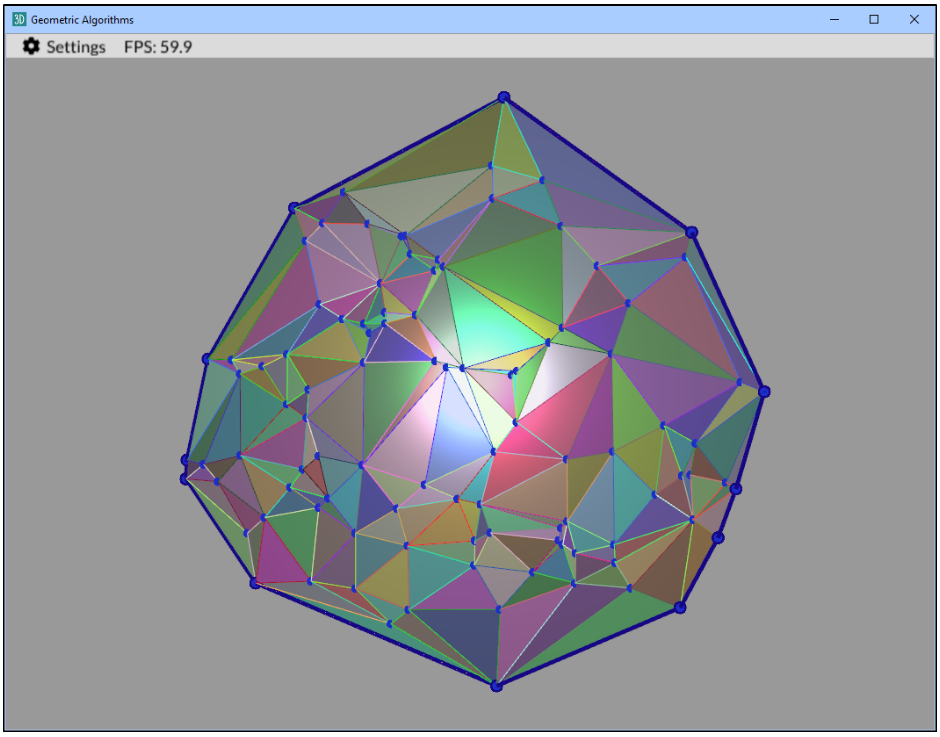

Figure 2: Point Cloud 2D to Convex Hull 2D (Delaunay Triangulation)

4. Convex Hull 3D

I've prepared two versions of the convex hull reconstruction from a point cloud.

The first is made from the random points uniformly distributed across a shepherd surface. Here is a randomization code:

```
inline glm::vec3 RandomUtilities::getUniformRandomInUnitSphereSurface() {
    glm::vec3 point;
    while(true) {
        const float theta = getUniformRandom(0.0f, 2.0f * glm::pi<float>());
        const float phi   = getUniformRandom(0.0f, glm::pi<float>());
        const auto  x = std::sqrt(1 - std::pow(std::cos(phi), 2)) * std::cos(theta);
        const auto  y = std::sqrt(1 - std::pow(std::cos(phi), 2)) * std::sin(theta);
        const float z = std::cos(phi);
        return {x, y, z};
    }
}
```

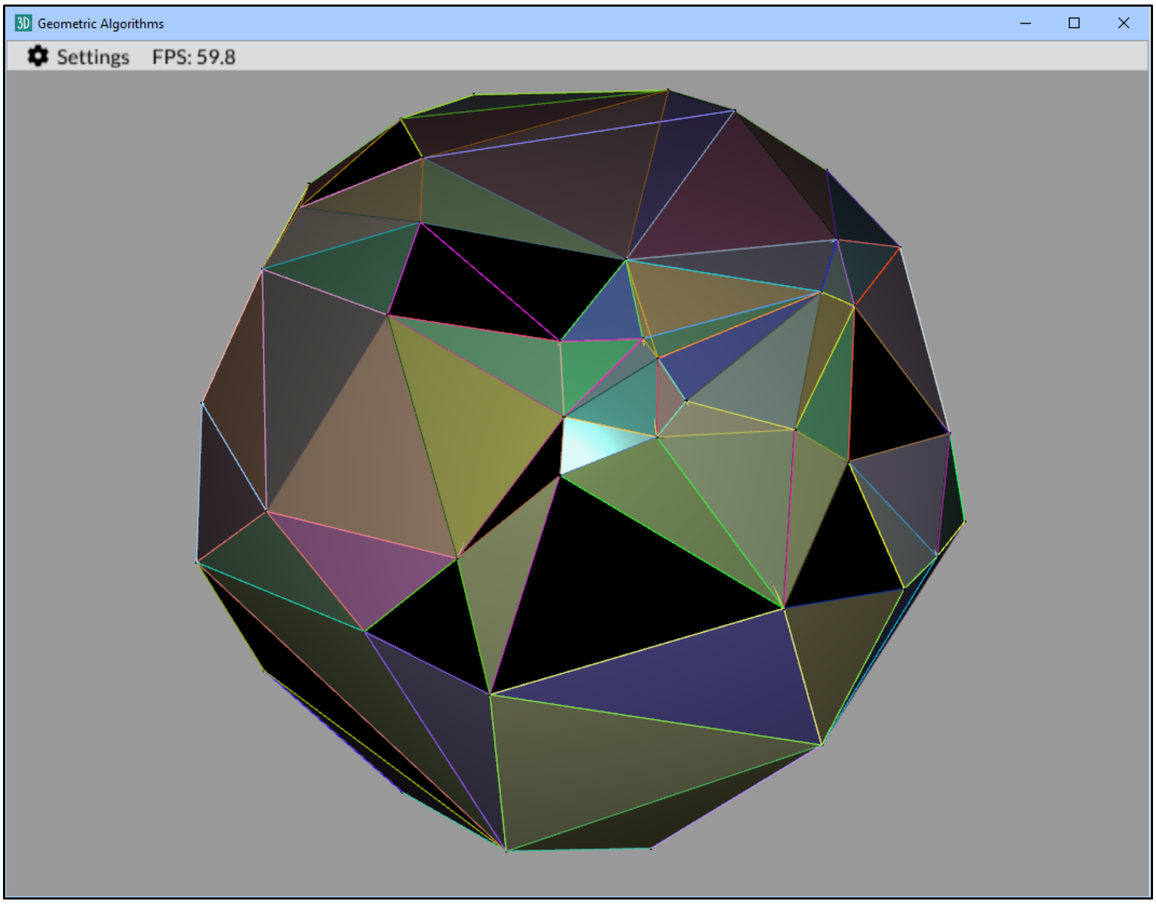

Figure 3: Point Cloud 3D (randomized on a sphere surface) to Convex Hull 3D

 Dmitry Ivanov (divanov@correo.ugr.es)

The second is made from the random points uniformly distributed within a sphere.

```
inline glm::vec3 RandomUtilities::getUniformRandomInUnitSphere() {
    glm::vec3 point;
    while(true) {
        point =
            glm::vec3(getUniformRandom(-1.0f, 1.0f),getUniformRandom(-1.0f, 1.0f), getUniformRandom(-1.0
        if(length2(point) >= 1)
            continue;
        return point;
    }
}
```

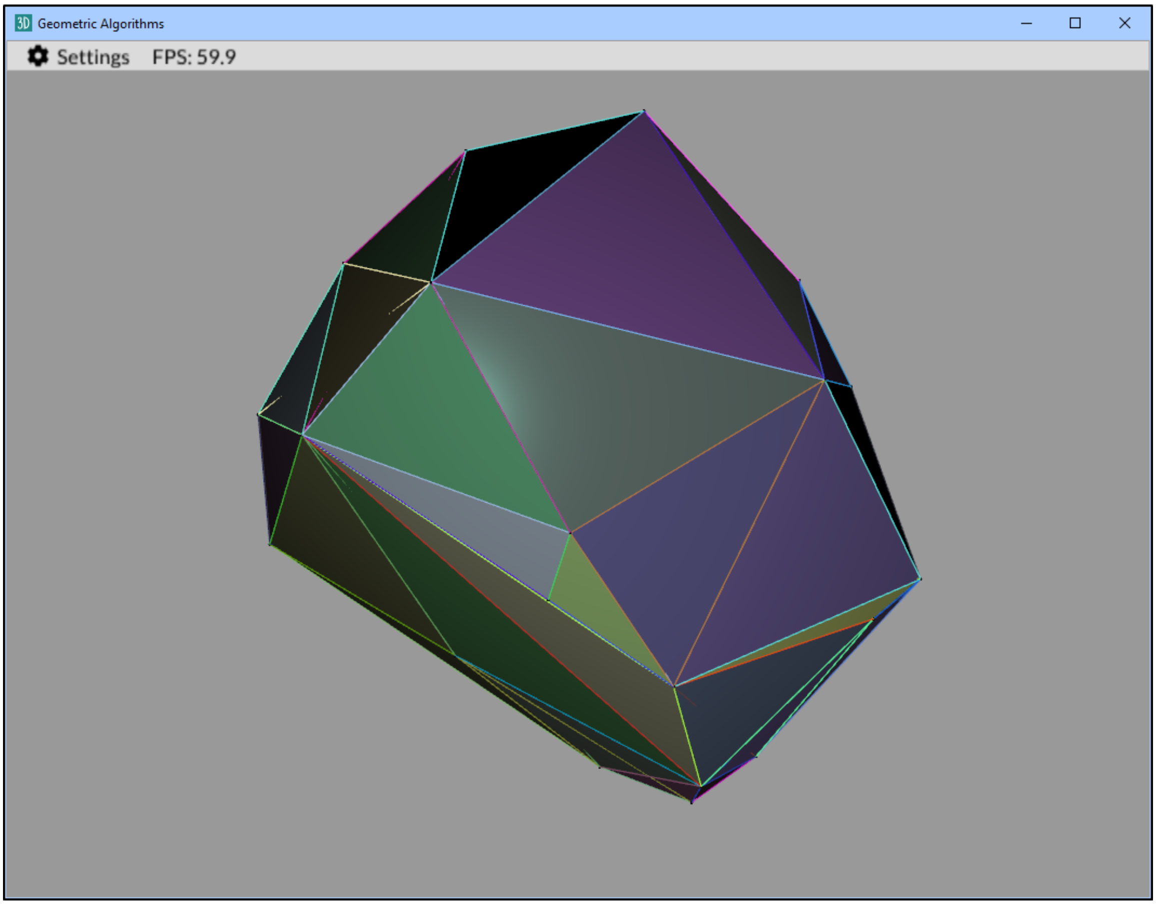

Figure 4: Point Cloud 3D (randomized in a sphere) to Convex Hull 3D

Dmitry Ivanov (divanov@correo.ugr.es)

5. Convex Hull from Triangular Mesh

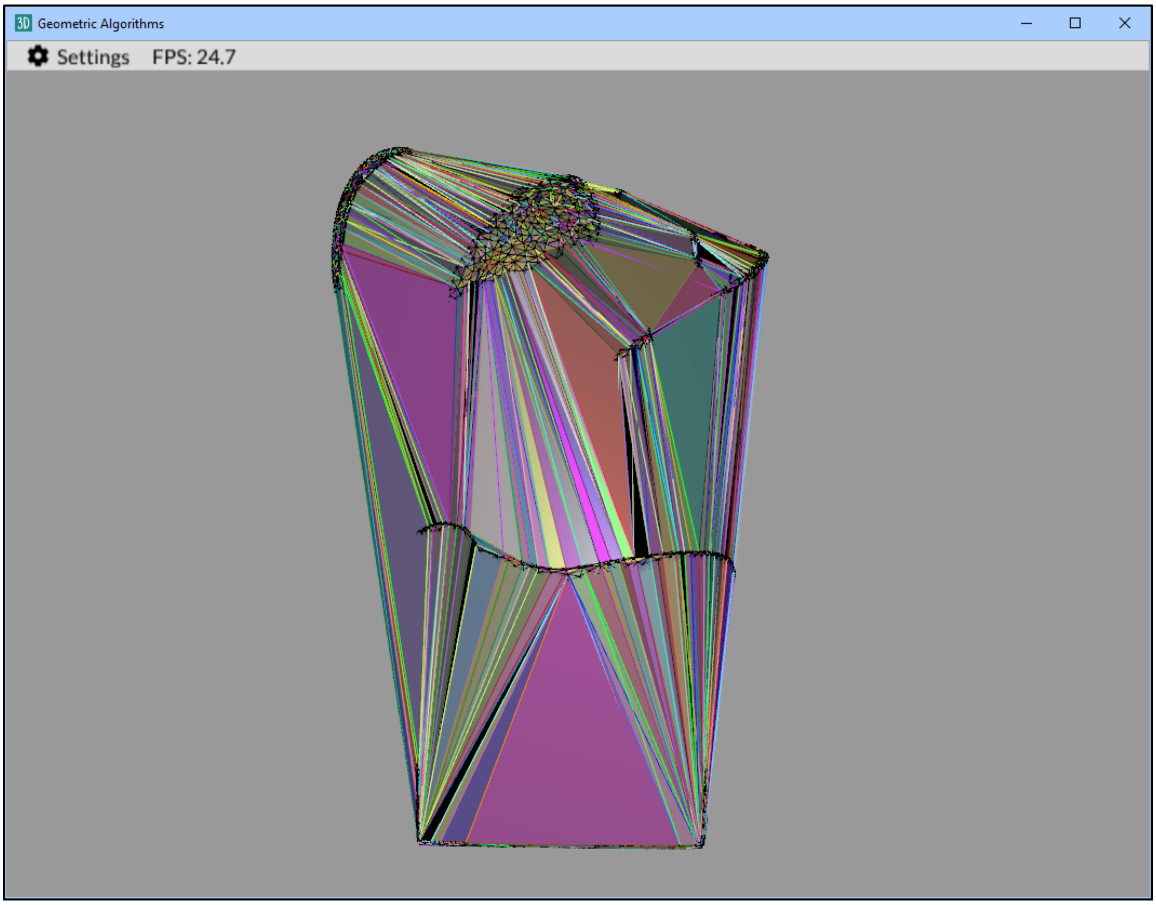

Figure 5: Convex Hull 3D generated from the Ajax model