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.

2. 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

3. 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

4. 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:

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

  // Practice 1:
  static void p1PointClouds(SceneContent& sc, int numPointClouds, int pointsPerCloud,
                            float scaleFactor, std::vector<Point>& randomPointsFromCloud,
                            std::vector<Point>& extremumPointInCloud);
  static void p1Lines(SceneContent& sc, const std::vector<Point>& randomPointsFromCloud);
  static void p1Polygon(SceneContent& sc, const std::vector<Point>& extremumPointInCloud);
  static void p1Bezier(SceneContent& sc, bool randomPoints = false, size_t pointNum = 4);
  static void p1Intersections(SceneContent& sc);
  static void p1All(SceneContent& sc);

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

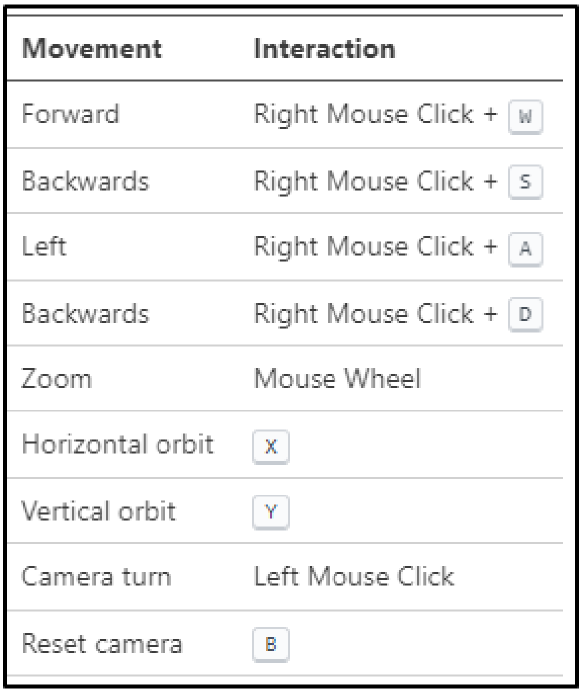
These methods are used in SceneContent:

```cpp
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:
  // Scenes::p1PointClouds(*this, numPointClouds, pointsPerCloud, scaleFactor,
  //                       randomPointsFromCloud, extremumPointInCloud);
  // Scenes::p1Lines(*this, randomPointsFromCloud);
  // Scenes::p1Polygon(*this, extremumPointInCloud);
  // Scenes::p1Bezier(*this);
  // Scenes::p1Bezier(*this, true, 5);
  // Scenes::p1Intersections(*this);

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

5. Practice 2.a

a. Point Cloud 3D

In this release, I've switched the camera from 2D to 3D, to allow the following 3D movements:



Movement	Interaction
Forward	Right Mouse Click + W
Backwards	Right Mouse Click + S
Left	Right Mouse Click + A
Backwards	Right Mouse Click + D
Zoom	Mouse Wheel
Horizontal orbit	X
Vertical orbit	Y
Camera turn	Left Mouse Click
Reset camera	B

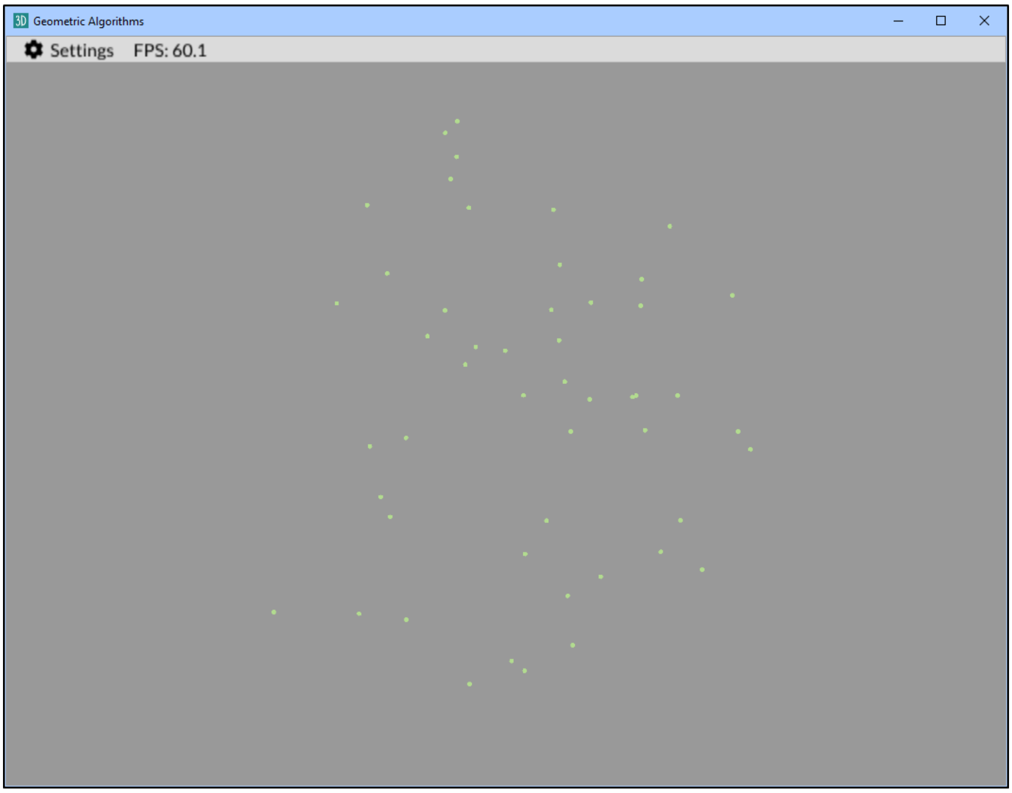


Figure 2: Point cloud 3D.

b. Two Lines, Ray and Segment

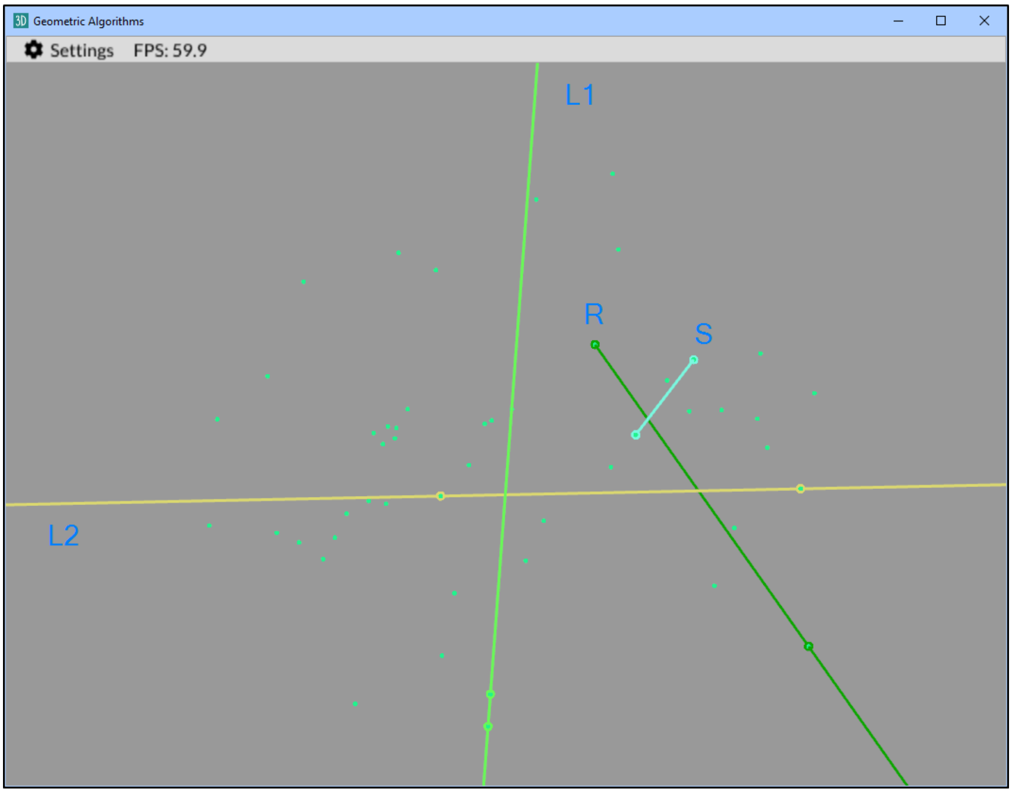


Figure 3: Two Lines, Ray and Segment.

There are two checks for lines parallelism and perpendicularity. Both checks go to the standard output.

```
==================================================
L1 and L2 are not perpendicular
L1 and L2 are not parallel
==================================================
L1 and L2 are not perpendicular
L1 and L2 are parallel
==================================================
L1 and L2 are perpendicular
L1 and L2 are not parallel
==================================================
```

c. The furthest point from the Segment

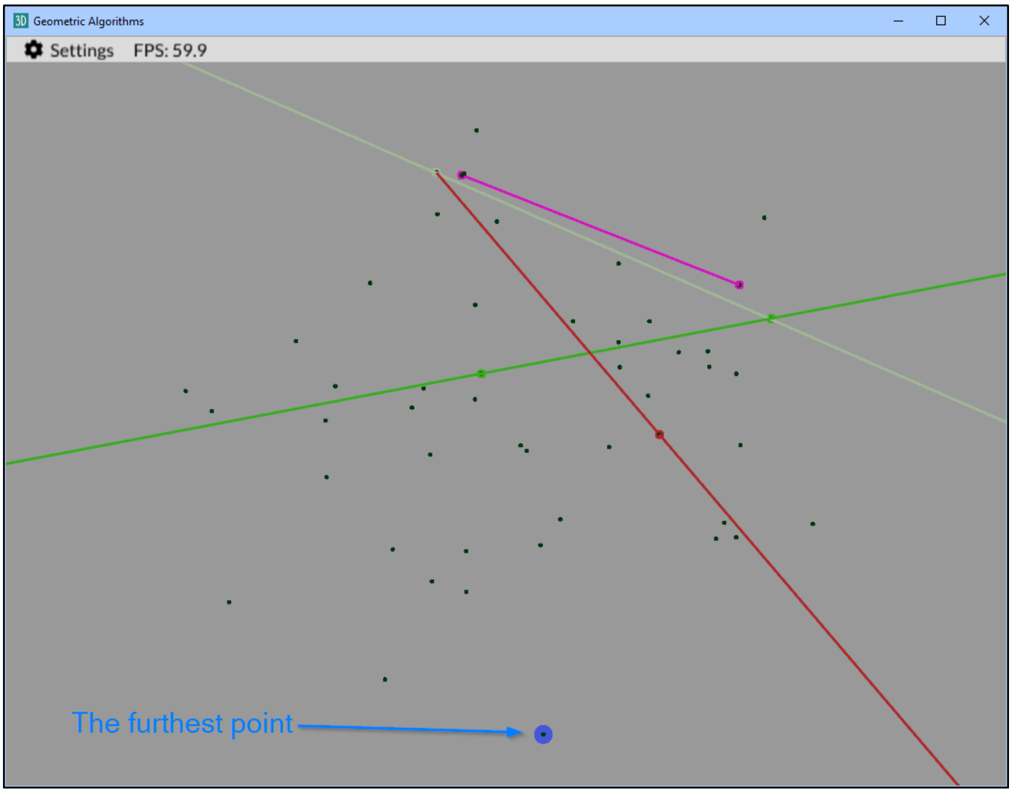
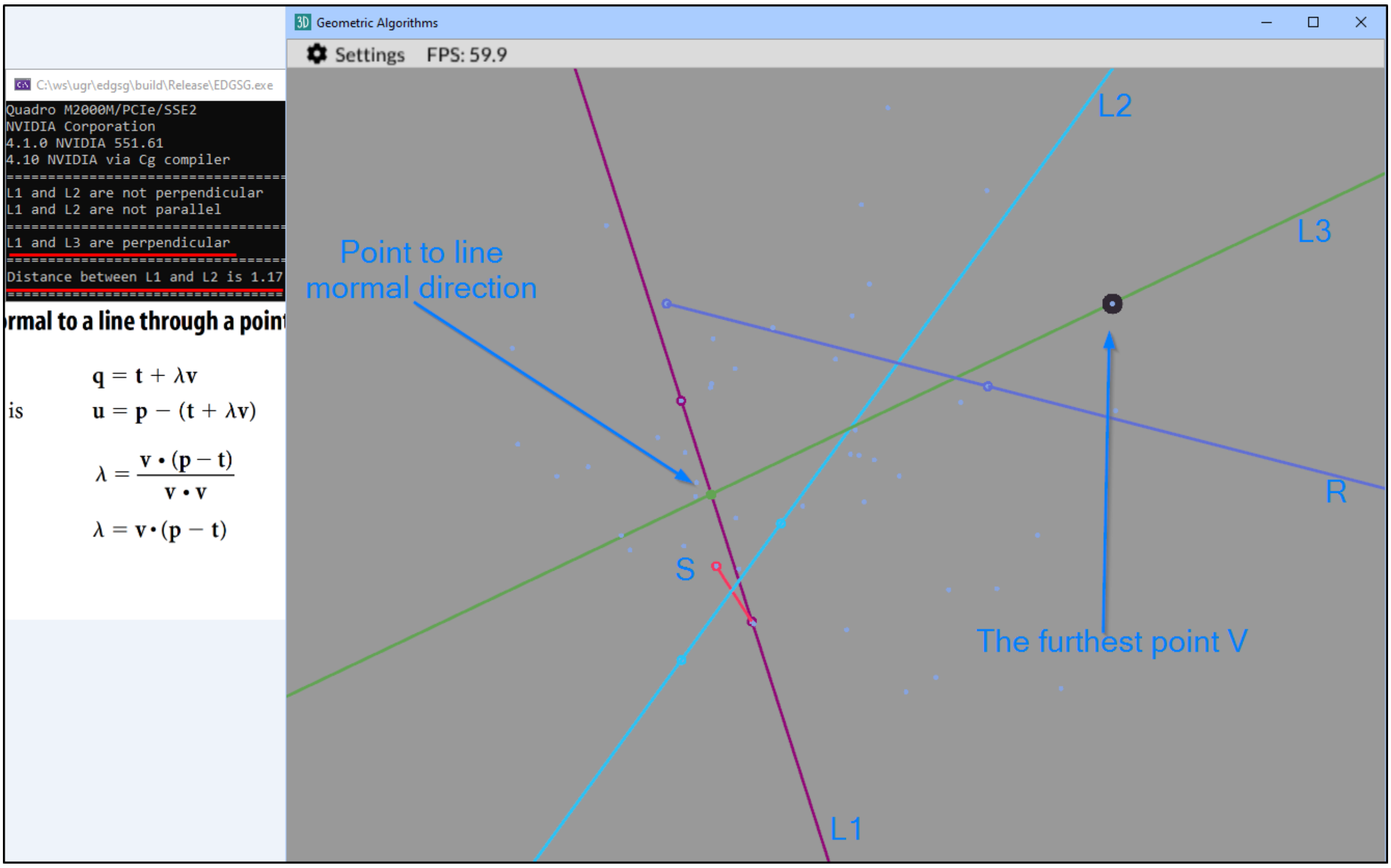


Figure 4: The furthest point from the Segment.

```
double maxDistance = 0;
Vect3d theMostDistantPoint;
auto sLine = new Line3d(s->getOrigin(), s->getDestination());
for(auto& point : pointCloud->getPoints())
{
    double distance = sLine->distance(point);
    if(distance > maxDistance)
    {
        maxDistance = distance;
        theMostDistantPoint = point;
    }
}
this.addNewModel((new DrawPoint(theMostDistantPoint))->setPointSize(20.0f));
```

d. Perpendicular line through the furthest point from the Segment



C:\ws\ugr\edgsg\build\Release\EDGSG.exe

```
Quadro M2000M/PCIe/SSE2
NVIDIA Corporation
4.1.0 NVIDIA 551.61
4.10 NVIDIA via Cg compiler
======================================
L1 and L2 are not perpendicular
L1 and L2 are not parallel
======================================
L1 and L3 are perpendicular
======================================
Distance between L1 and L2 is 1.17
======================================
```

rmal to a line through a point

$$\mathbf{q} = \mathbf{t} + \lambda\mathbf{v}$$

is $$\mathbf{u} = \mathbf{p} - (\mathbf{t} + \lambda\mathbf{v})$$

$$\lambda = \frac{\mathbf{v} \bullet (\mathbf{p} - \mathbf{t})}{\mathbf{v} \bullet \mathbf{v}}$$

$$\lambda = \mathbf{v} \bullet (\mathbf{p} - \mathbf{t})$$

Figure 5: Perpendicular line through the furthest point from the Segment.
Distance between L1 and L2.

e. AABB from the Point Cloud

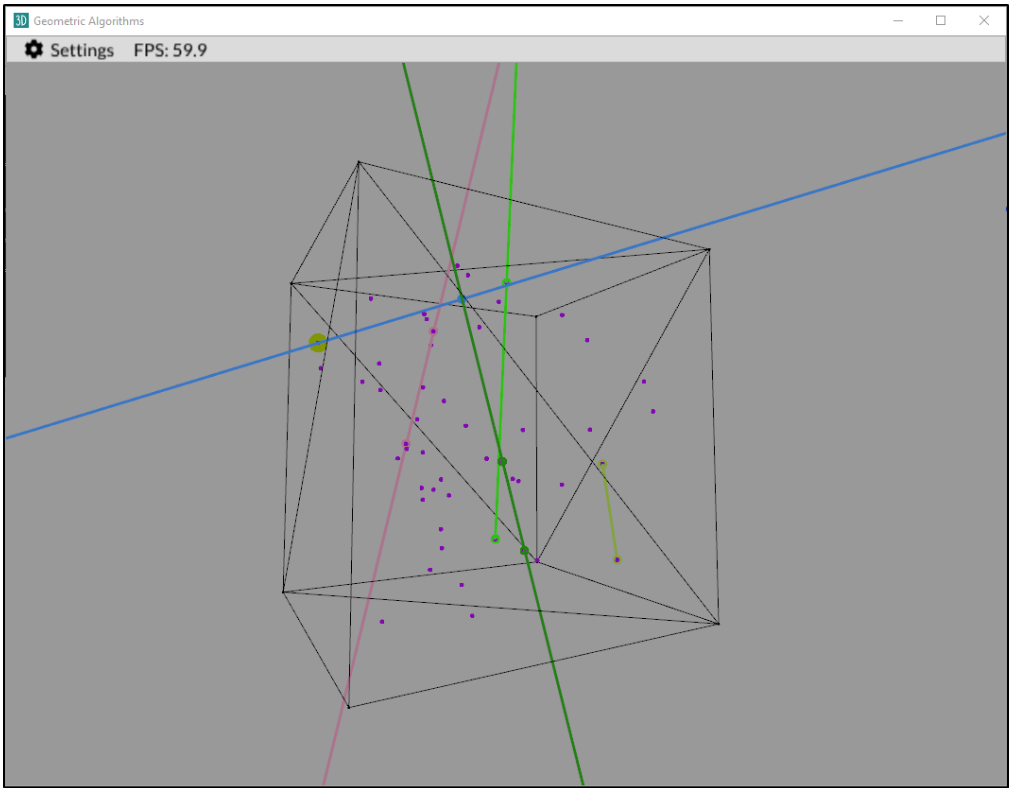
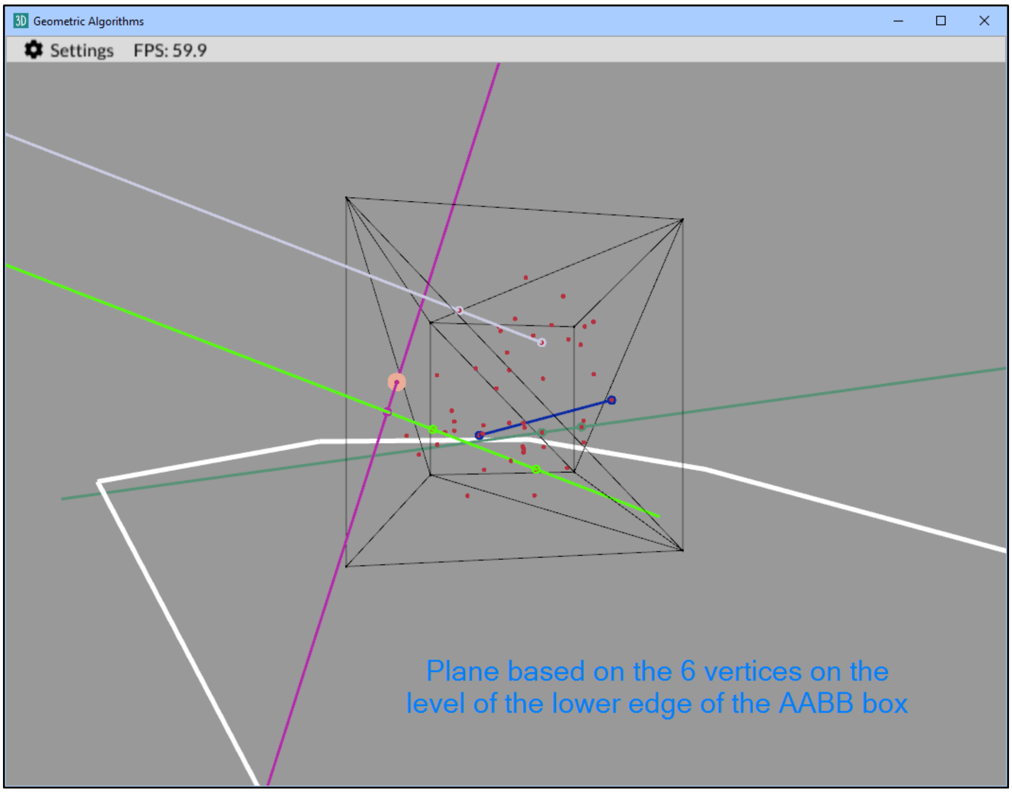


Figure 6: AABB box from the Point Cloud

There are two important problems in providing PointCloud3d class. **The default _maxPoint value is defined by positive infinity**, which makes it impossible to calculate the updated maximum and minimum for newly added points. This should be fixed using the following patch:

```
diff --git c/Source/Geometry/PointCloud3d.cpp w/Source/Geometry/PointCloud3d.cpp
index e5750c2..c4a7781 100644
--- c/Source/Geometry/PointCloud3d.cpp
+++ w/Source/Geometry/PointCloud3d.cpp
@@ -7,14 +7,14 @@
 PointCloud3d::PointCloud3d()
-     : _maxPoint(INFINITY, -INFINITY, -INFINITY)
+     : _maxPoint(-INFINITY, -INFINITY, -INFINITY)
      , _minPoint(INFINITY, INFINITY, INFINITY)
 {
 }
PointCloud3d::PointCloud3d(std::vector<Vect3d>& pointCloud)
      : _points(pointCloud)
-     , _maxPoint(INFINITY, -INFINITY, -INFINITY)
+     , _maxPoint(-INFINITY, -INFINITY, -INFINITY)
      , _minPoint(INFINITY, INFINITY, INFINITY)
 {
 }
```

f. Plane on the lower edge of AABB



Plane based on the 6 vertices on the level of the lower edge of the AABB box

Figure 7: Plane on the lower edge of AABB.

6. Practice 2.b

a. Random Plane

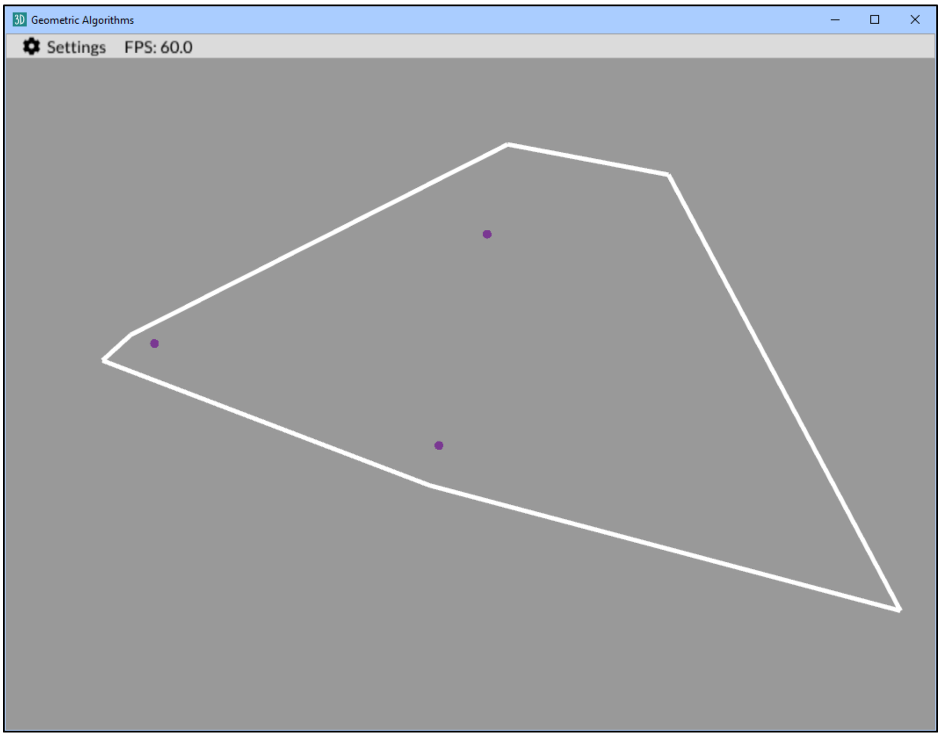


Figure 8: Random Plane.

b. Plane intersection

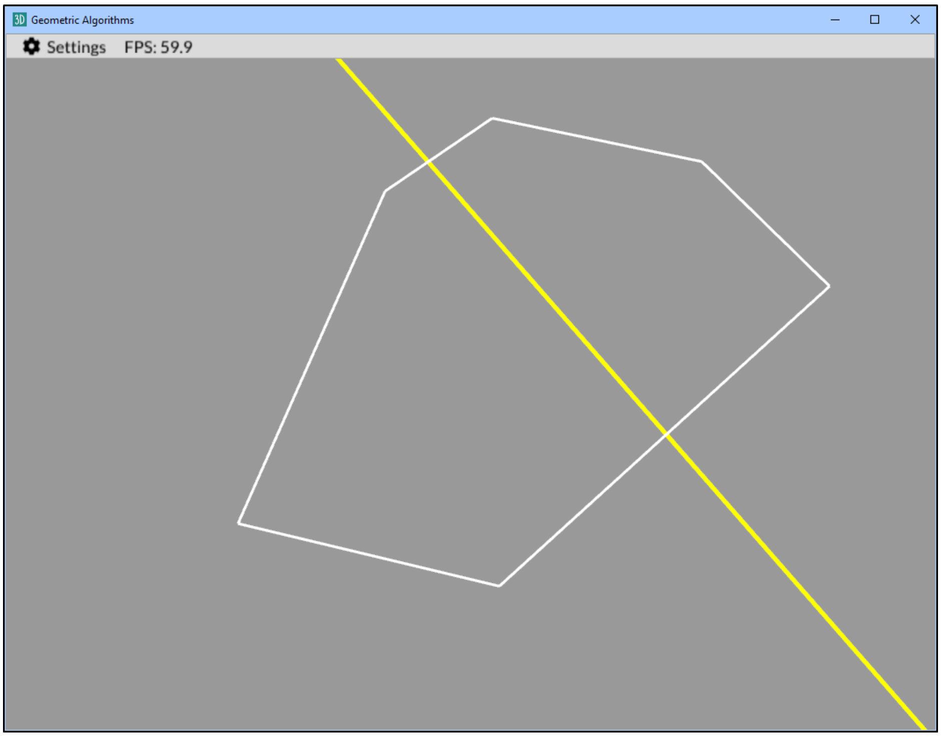
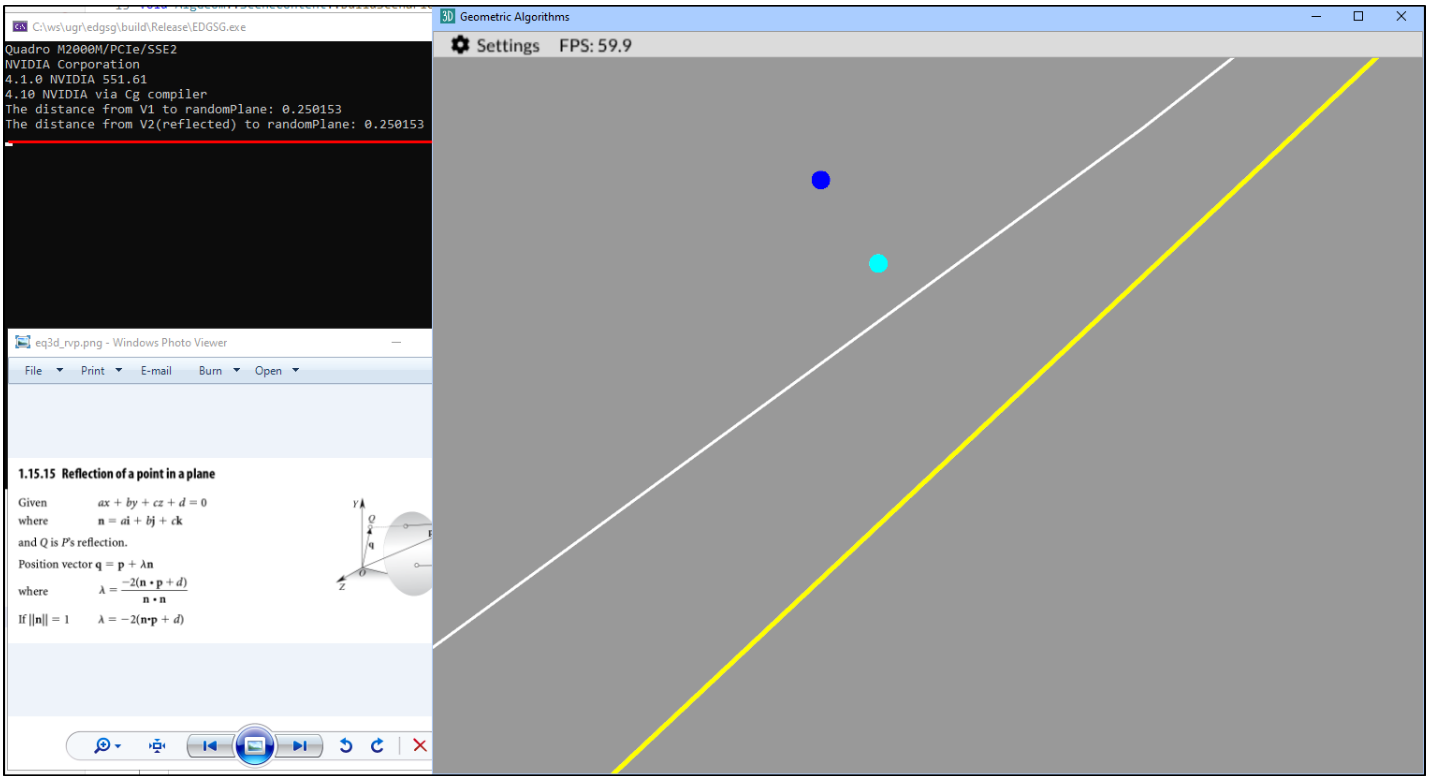


Figure 9: Yellow line painted on the intersection of the two planes.

c. Plane to point distance


C:\ws\ugr\edgsg\build\Release\EDGSG.exe
eq3d_rvp.png - Windows Photo Viewer
File
Print
E-mail
Burn
Open
Y
Q
q
P
O
Z
Geometric Algorithms
Settings
FPS: 59.9

```
Quadro M2000M/PCIe/SSE2
NVIDIA Corporation
4.1.0 NVIDIA 551.61
4.10 NVIDIA via Cg compiler
The distance from V1 to randomPlane: 0.250153
The distance from V2(reflected) to randomPlane: 0.250153
```

1.15.15 Reflection of a point in a plane

Given $ax + by + cz + d = 0$

where $\mathbf{n} = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$

and Q is P's reflection.

Position vector $\mathbf{q} = \mathbf{p} + \lambda\mathbf{n}$

where $\lambda = \frac{-2(\mathbf{n} \cdot \mathbf{p} + d)}{\mathbf{n} \cdot \mathbf{n}}$

If $\|\mathbf{n}\| = 1$ $\lambda = -2(\mathbf{n} \cdot \mathbf{p} + d)$

Figure 10: Distance to the blue point V1.

```
The distance from V1 to randomPlane: 0.250153
The distance from V2(reflected) to randomPlane: 0.250153
```

7. Practice 2.c

a. Triangular model

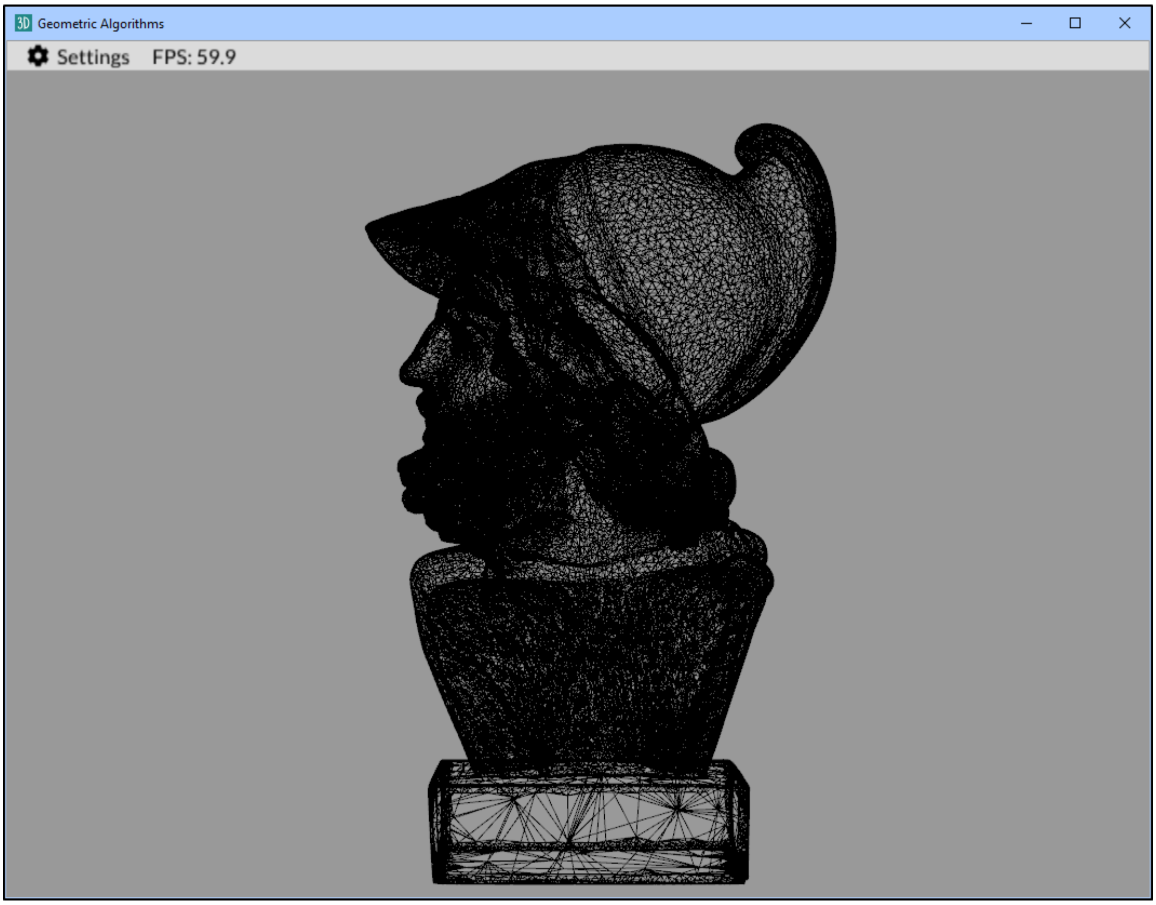


Figure 11: Ajax model in wireframe.

b. Six triangles, whose normals are orthogonal to the axis planes

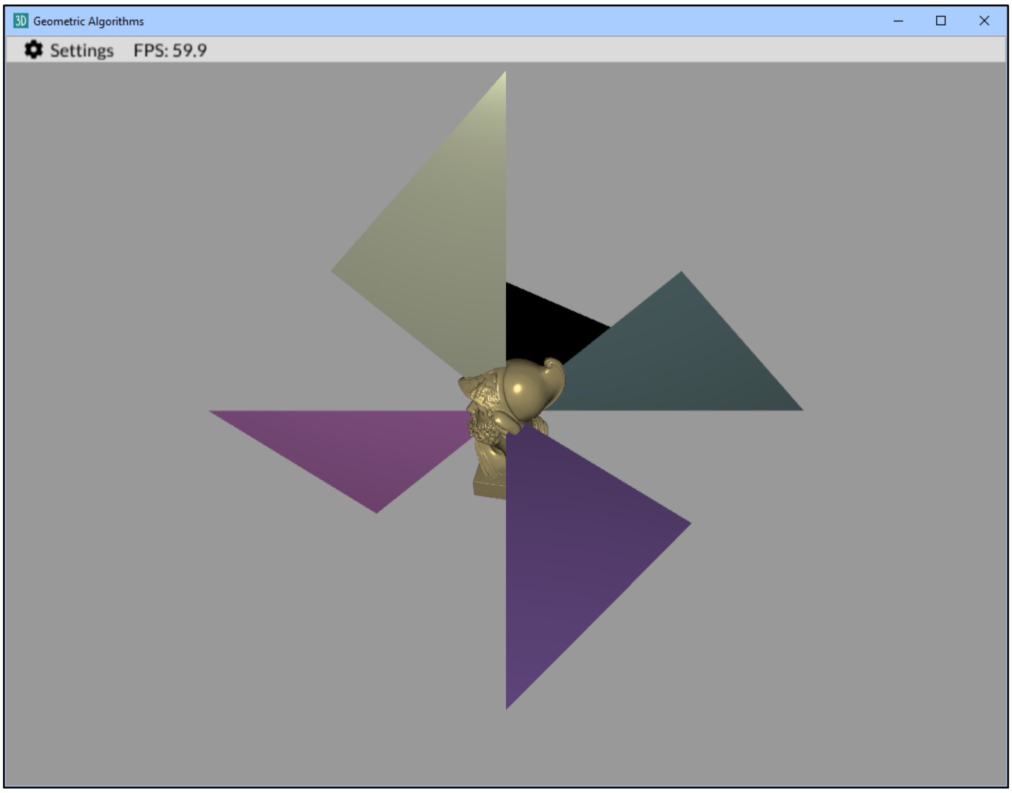


Figure 12: Front side.

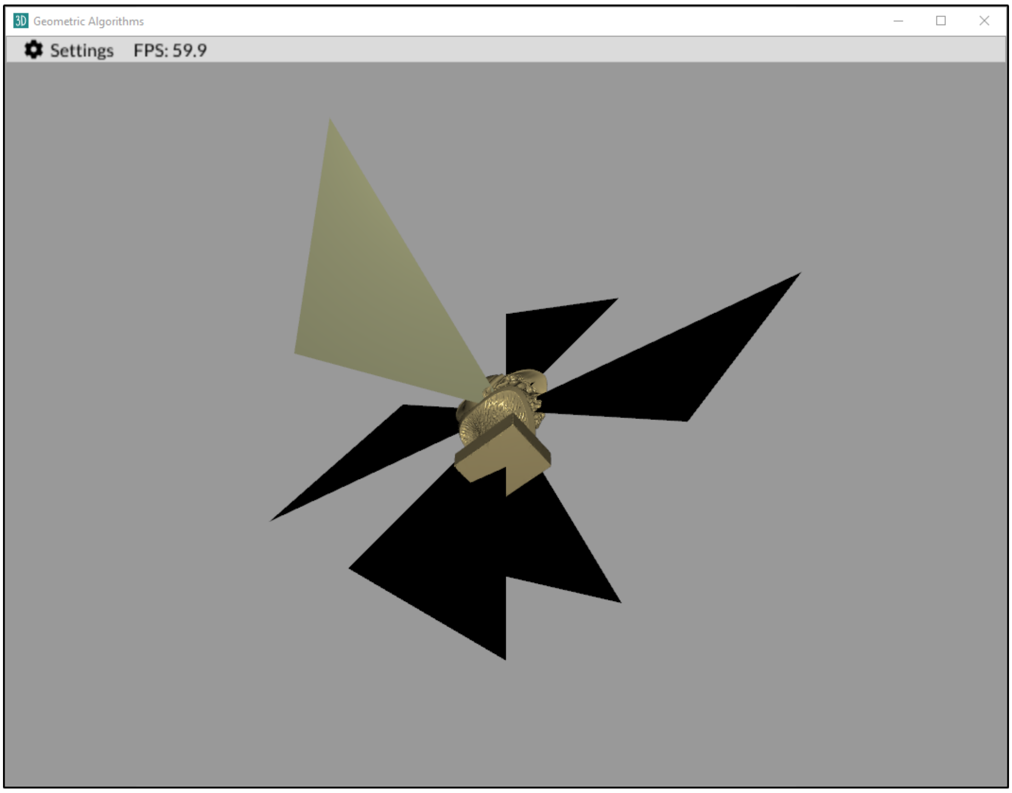


Figure 13: Back side.