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:
  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);

  // Practice 3:
  static void p3(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:
  // 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);

  // Scenes::p3(*this);
}
```

3. Execution summary

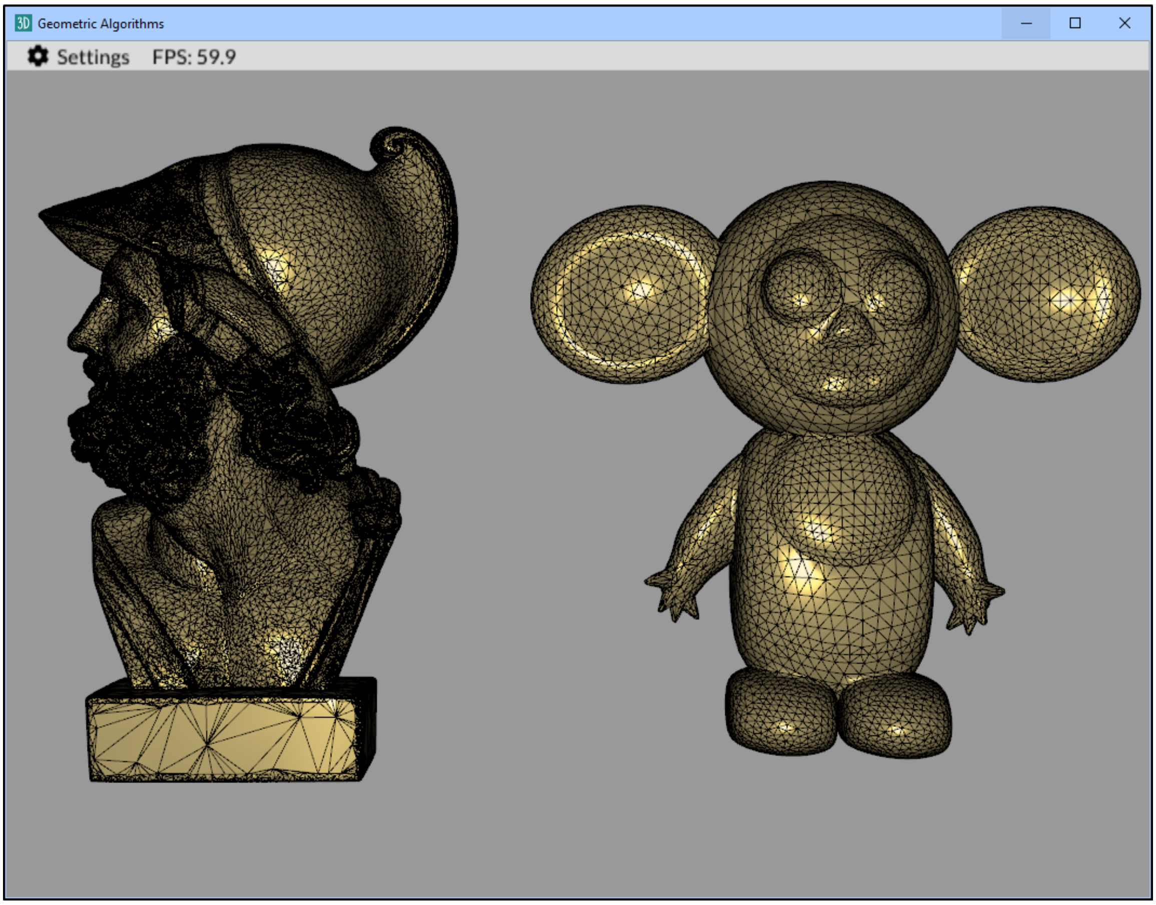

Figure 1: Sampled models

Model	Triangles	Brute Force	Line Sweep	AABB Sampling
Ajax	163368	231435 ms	12931 ms	54239 ms
Cheburashka	13334	12274 ms	951 ms	3144 ms

Execution time comparison.

4. Brute Force Voxelization

a. Ajax model

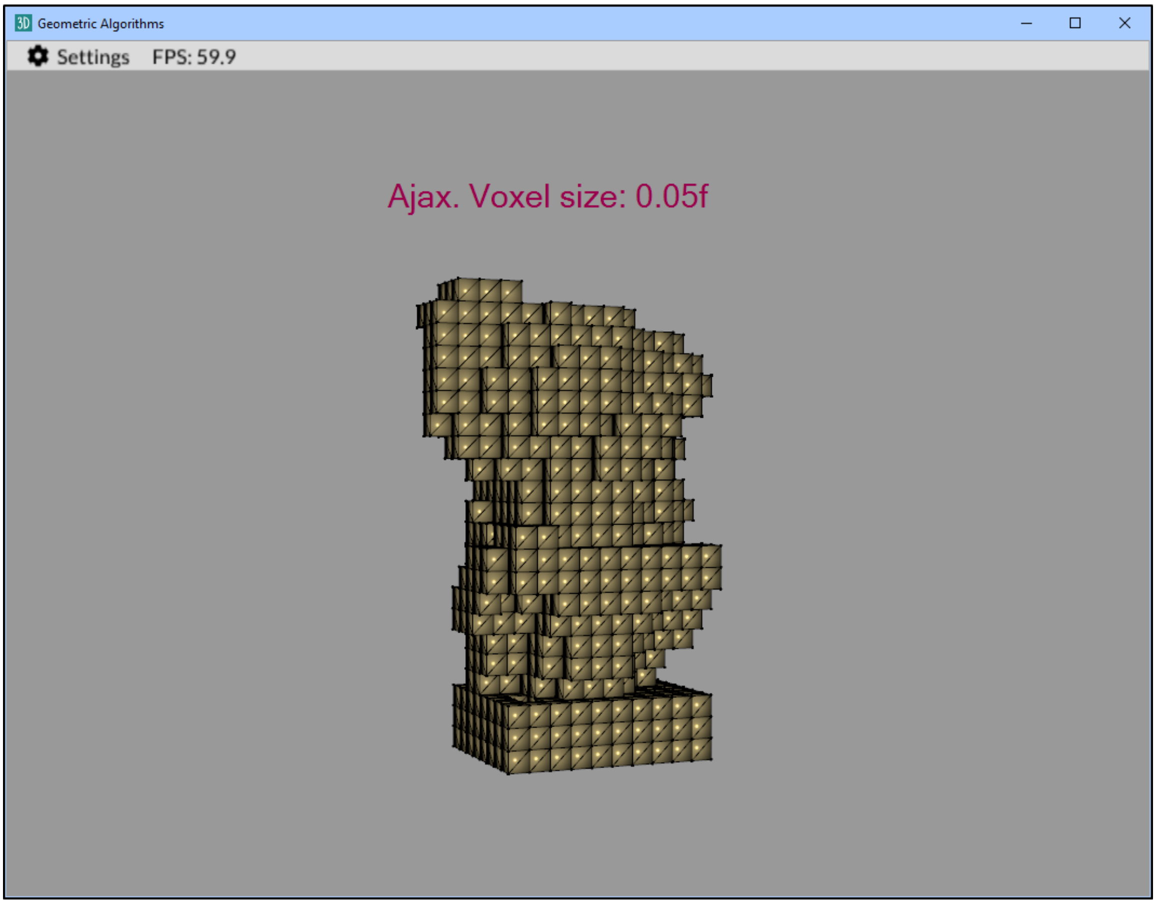

Figure 2: Ajax Brute Force Voxelization Result

```
==================================================
Voxelization (Brute Force) started
Voxelization (Brute Force) ended. Duration: 231435 ms
==================================================
```

Dmitry Ivanov (divanov@correo.ugr.es)

b. Cheburashka model

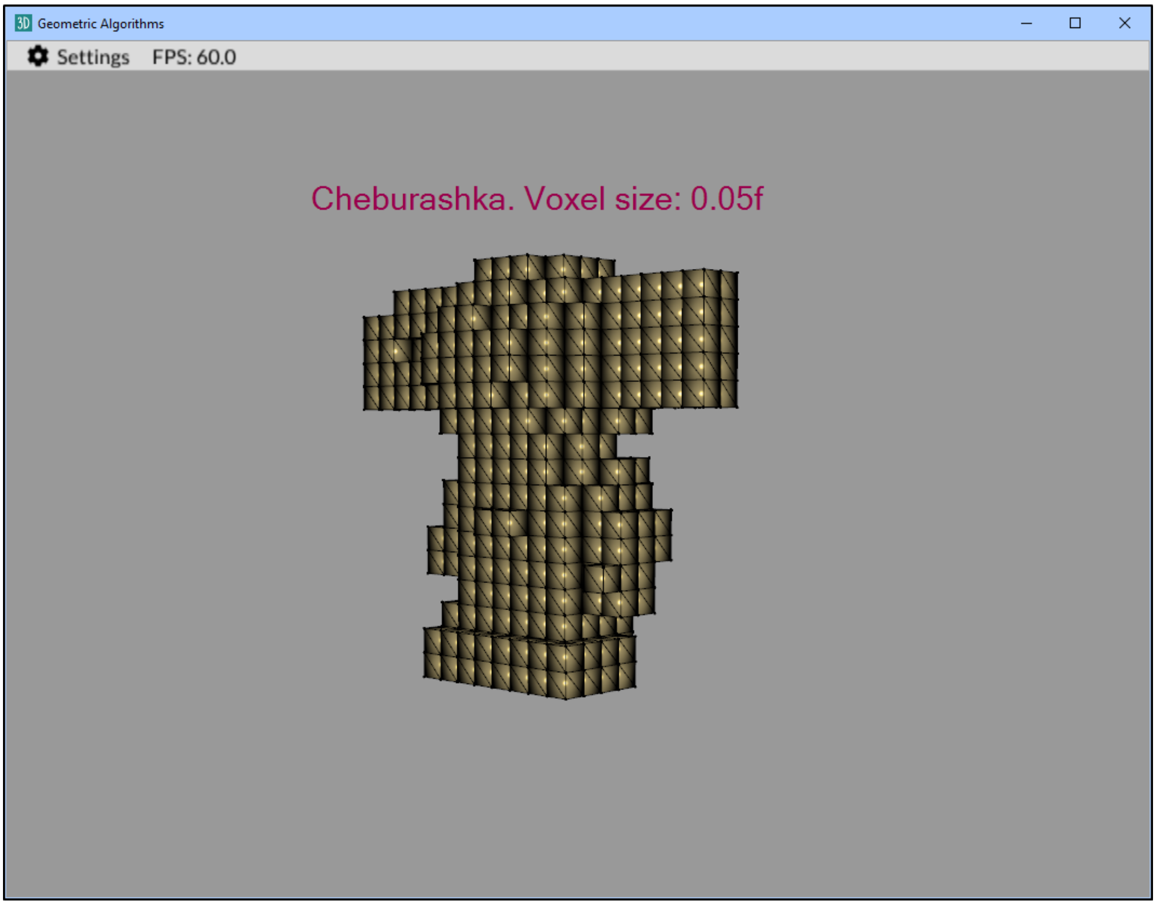

Figure 3: Cheburashka Brute Force Voxelization Result

```
==================================================
Voxelization (Brute Force) started
Voxelization (Brute Force) ended. Duration: 12274 ms
==================================================
```

Dmitry Ivanov (divanov@correo.ugr.es)

5. Line Sweep Voxelization

a. Ajax model

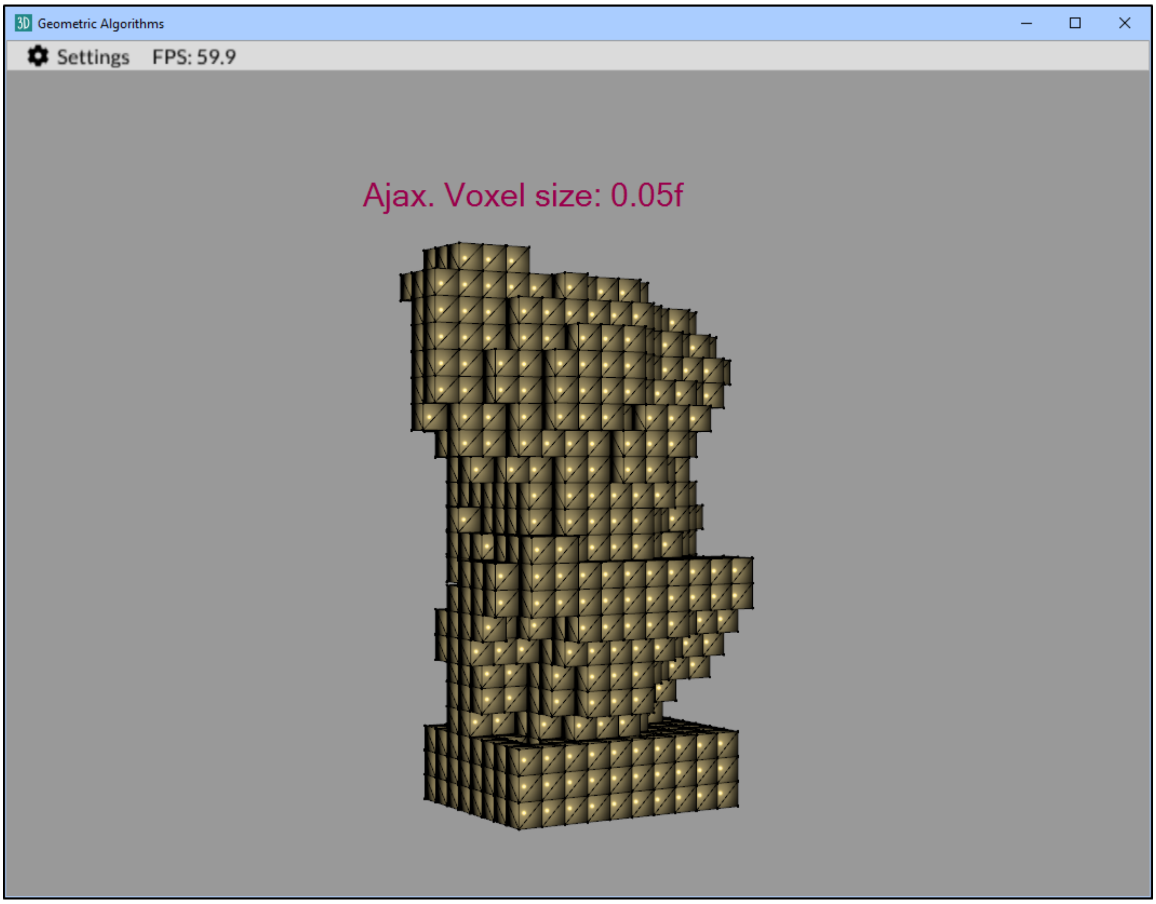

Figure 4: Ajax Line Sweep Voxelization Result

```
==================================================
Voxelization (Line Sweep) started
Voxelization (Line Sweep) ended. Duration: 12931 ms
==================================================
```

b. Cheburashka model

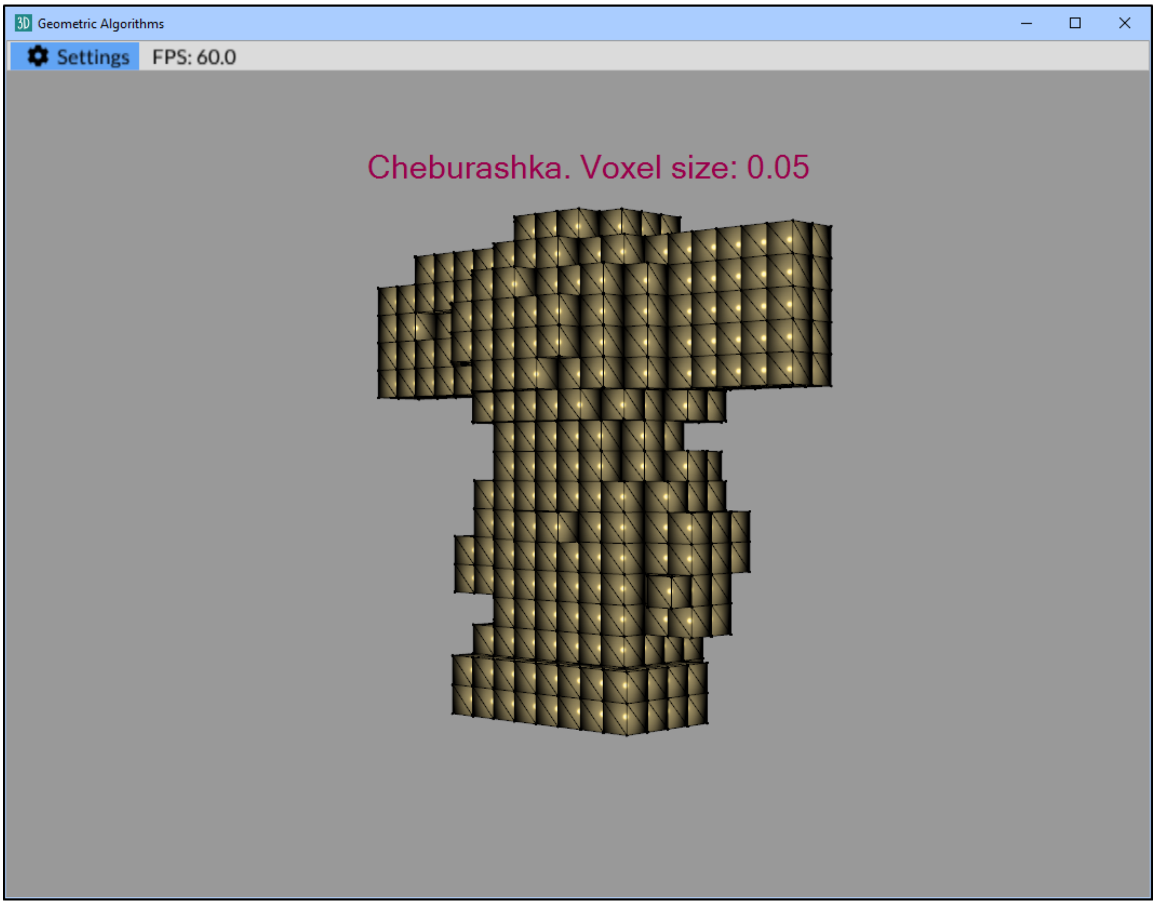

Figure 5: Cheburashka Line Sweep Voxelization Result

```
==================================================
Voxelization (Line Sweep) started
Voxelization (Line Sweep) ended. Duration: 951 ms
==================================================
```

6. AABB-Sampling Voxelization

a. Ajax model

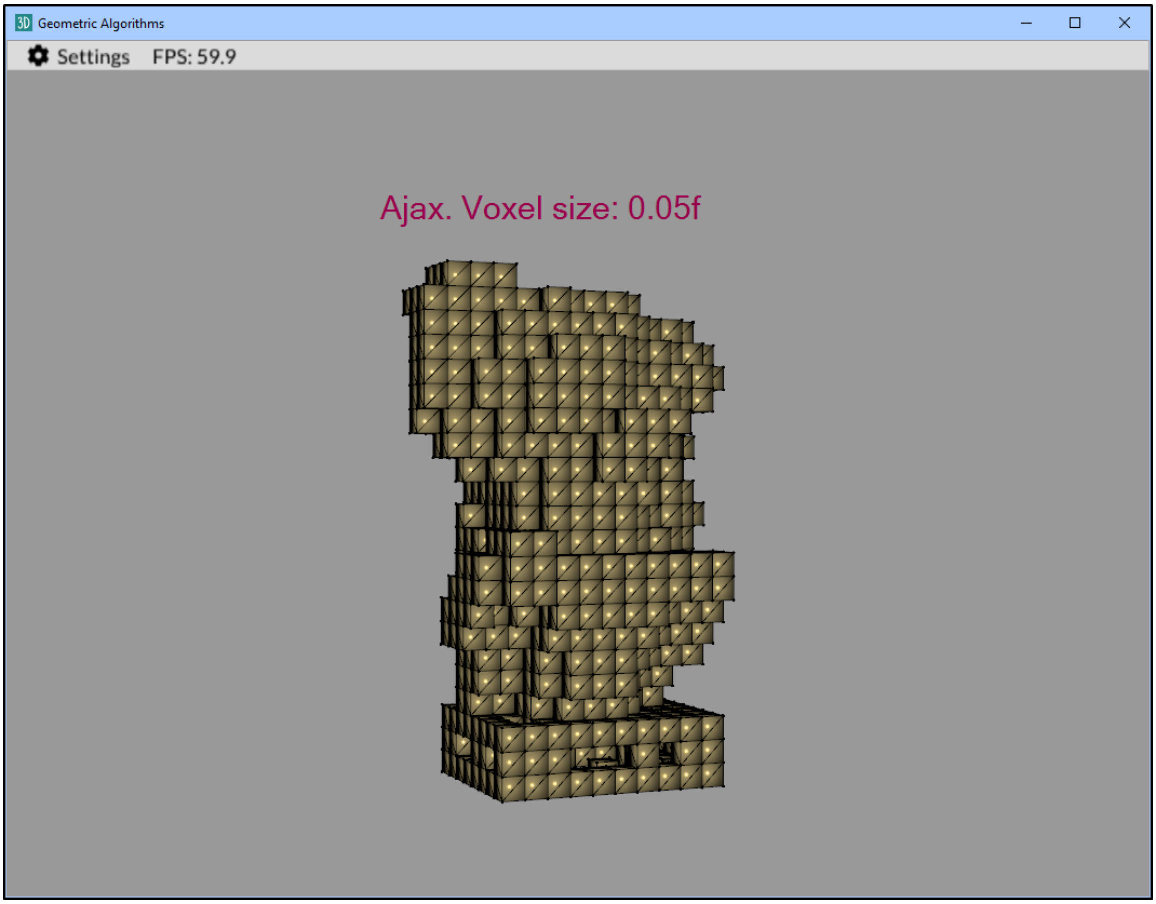

Figure 6: Ajax AABB-Sampling Voxelization Result

```
==================================================
Voxelization (AABB-Sampling) started
Voxelization (AABB-Sampling) ended. Duration: 54239 ms
==================================================
```

b. Cheburashka model

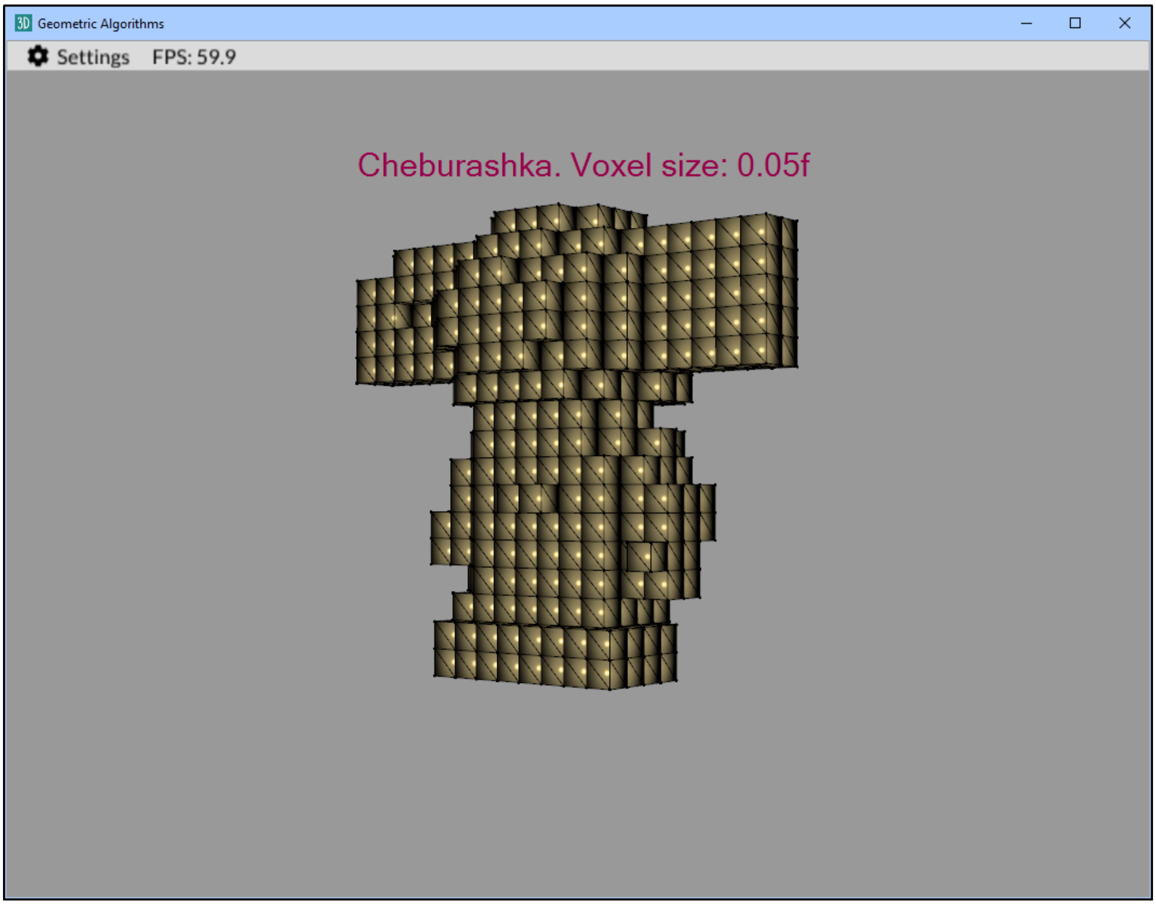

Figure 7: Cheburashka AABB-Sampling Voxelization Result

```
==================================================
Voxelization (AABB-Sampling) started
Voxelization (AABB-Sampling) ended. Duration: 3144 ms
==================================================
```