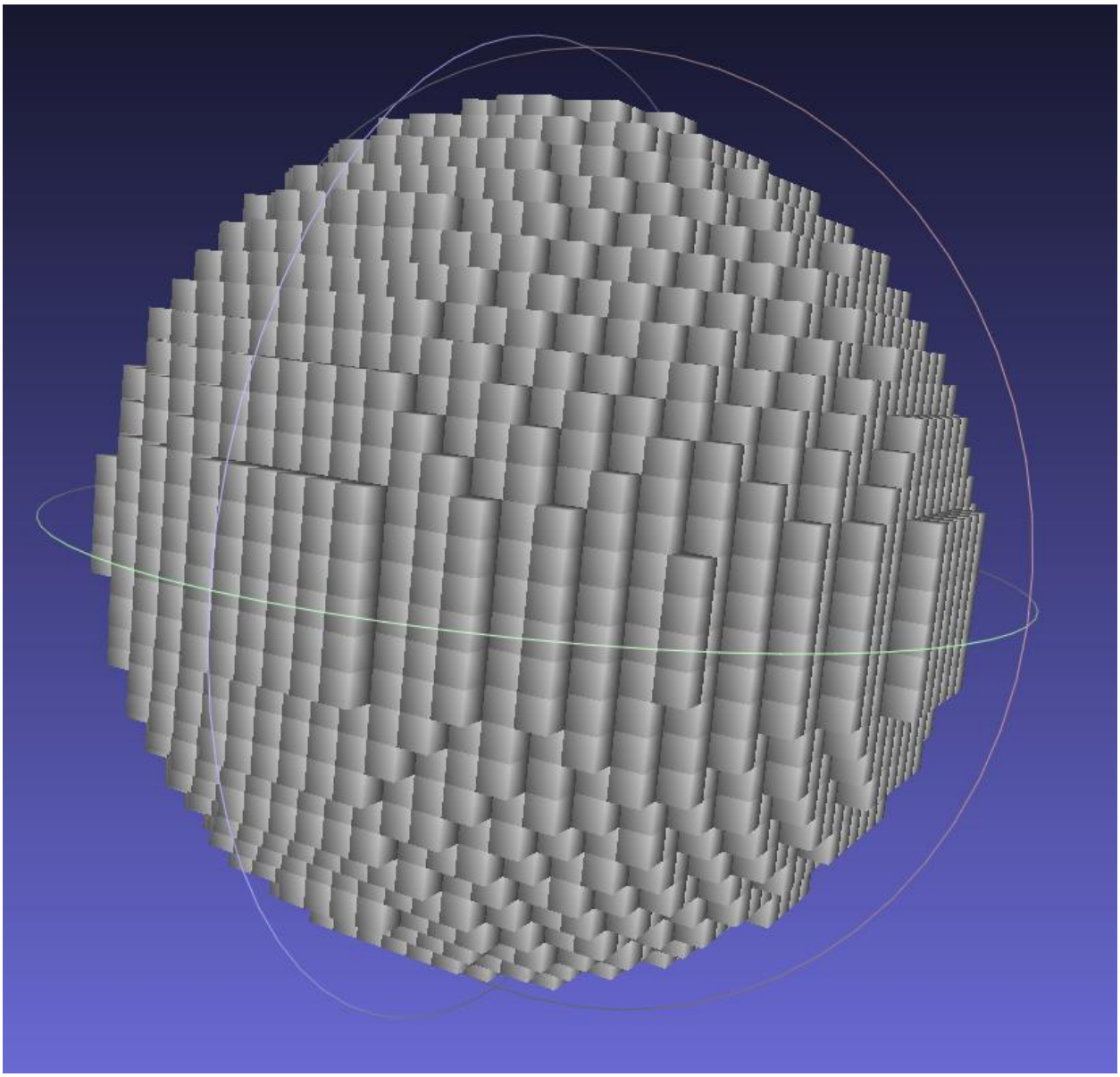
01.110 Computational Fabrication

Lab 1: 3D Scanning with Kinect

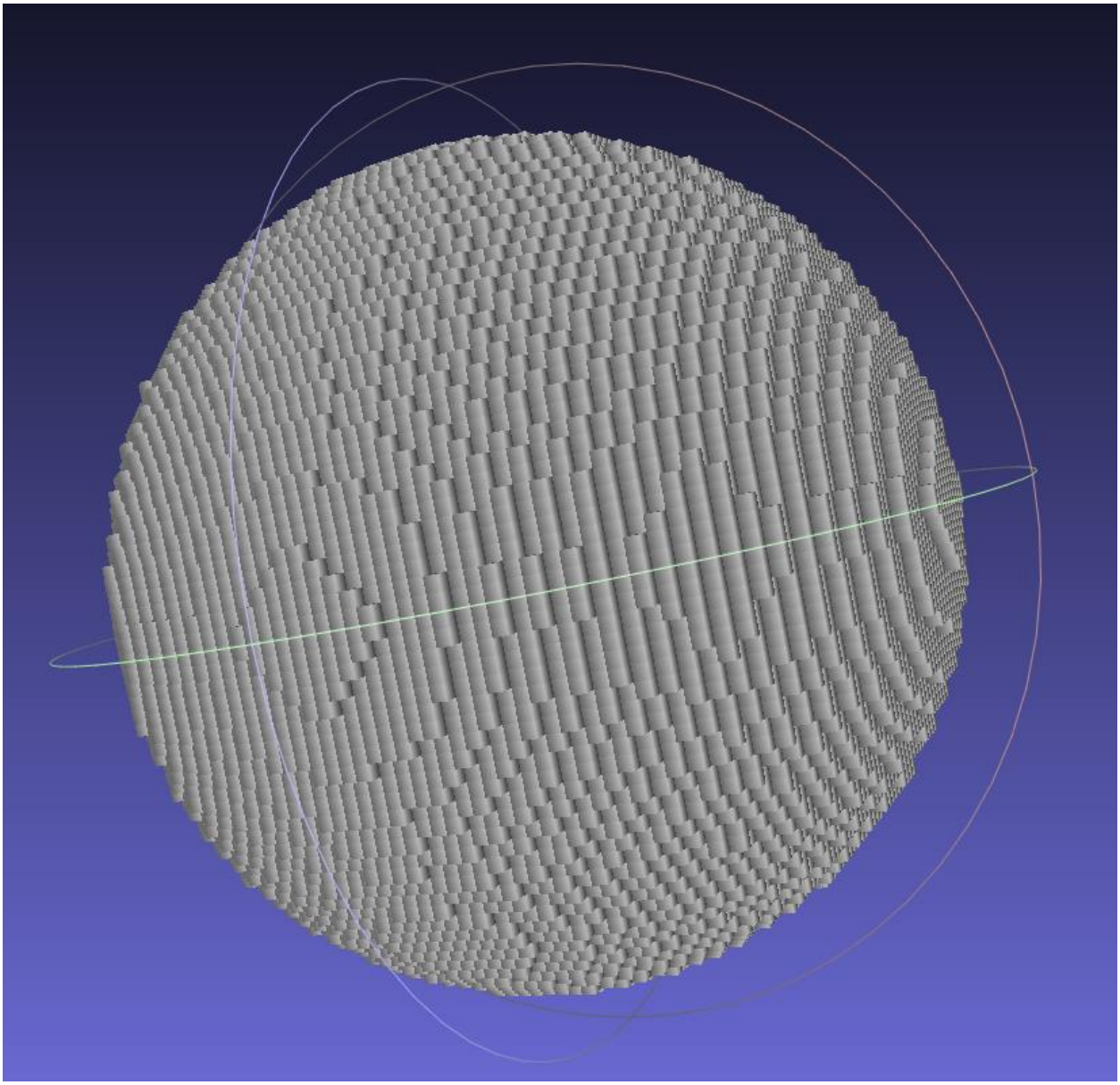
Zhang Zhexian (1001214)

Images of voxelizations of all example files at 32x32x32 resolution and 64x64x64 resolution

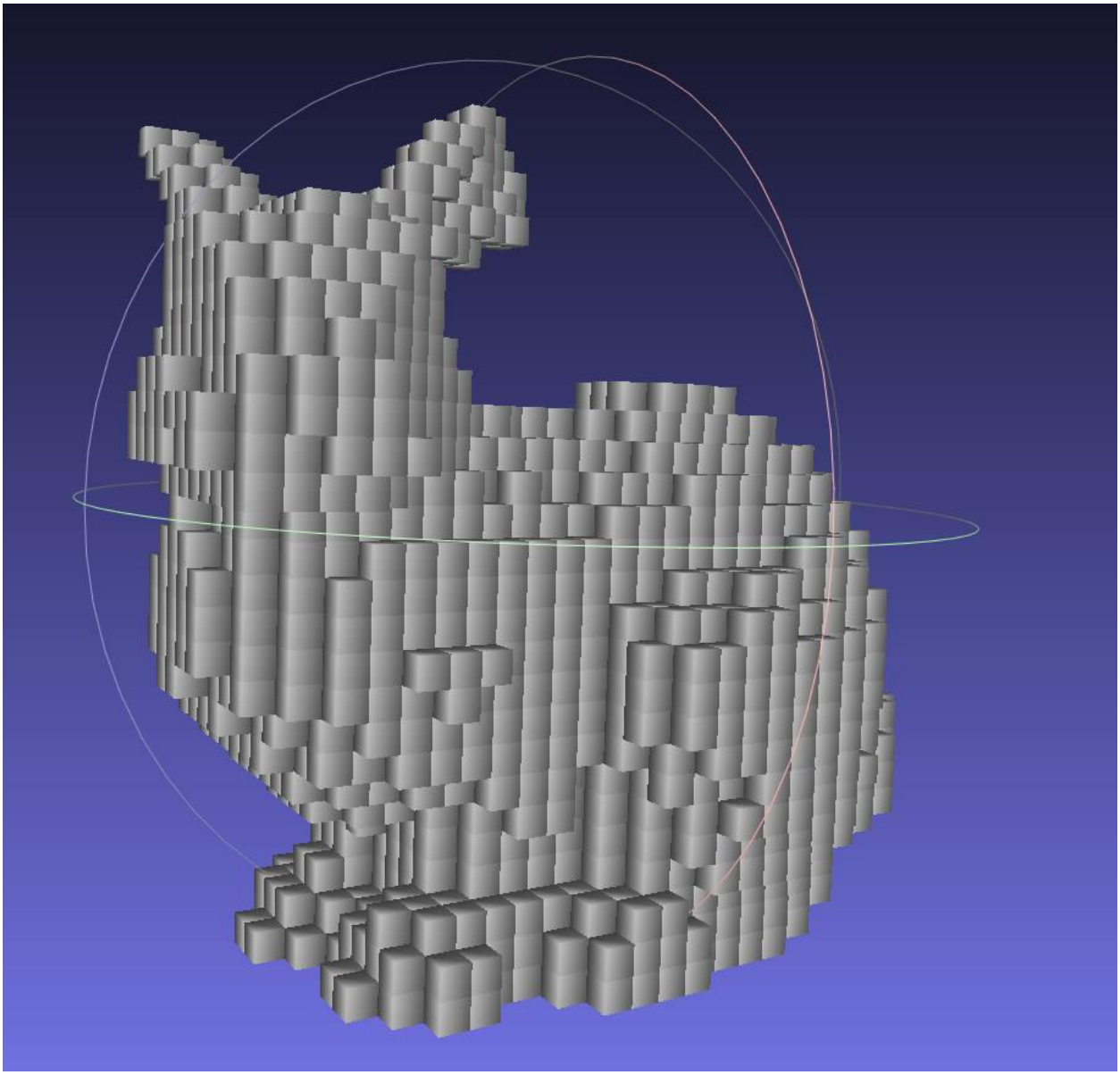
Sphere (32x32x32):



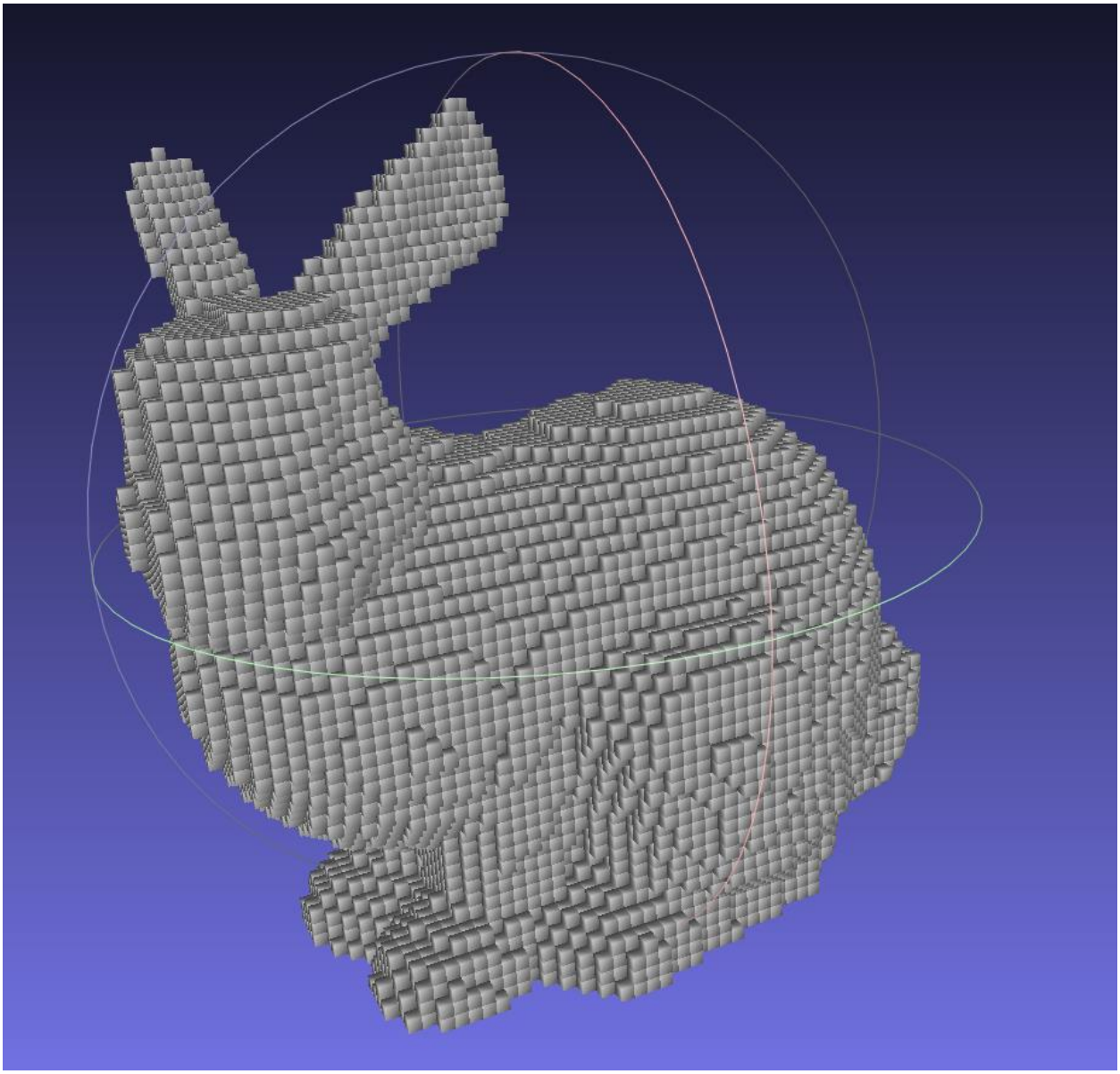
Sphere (64x64x64):



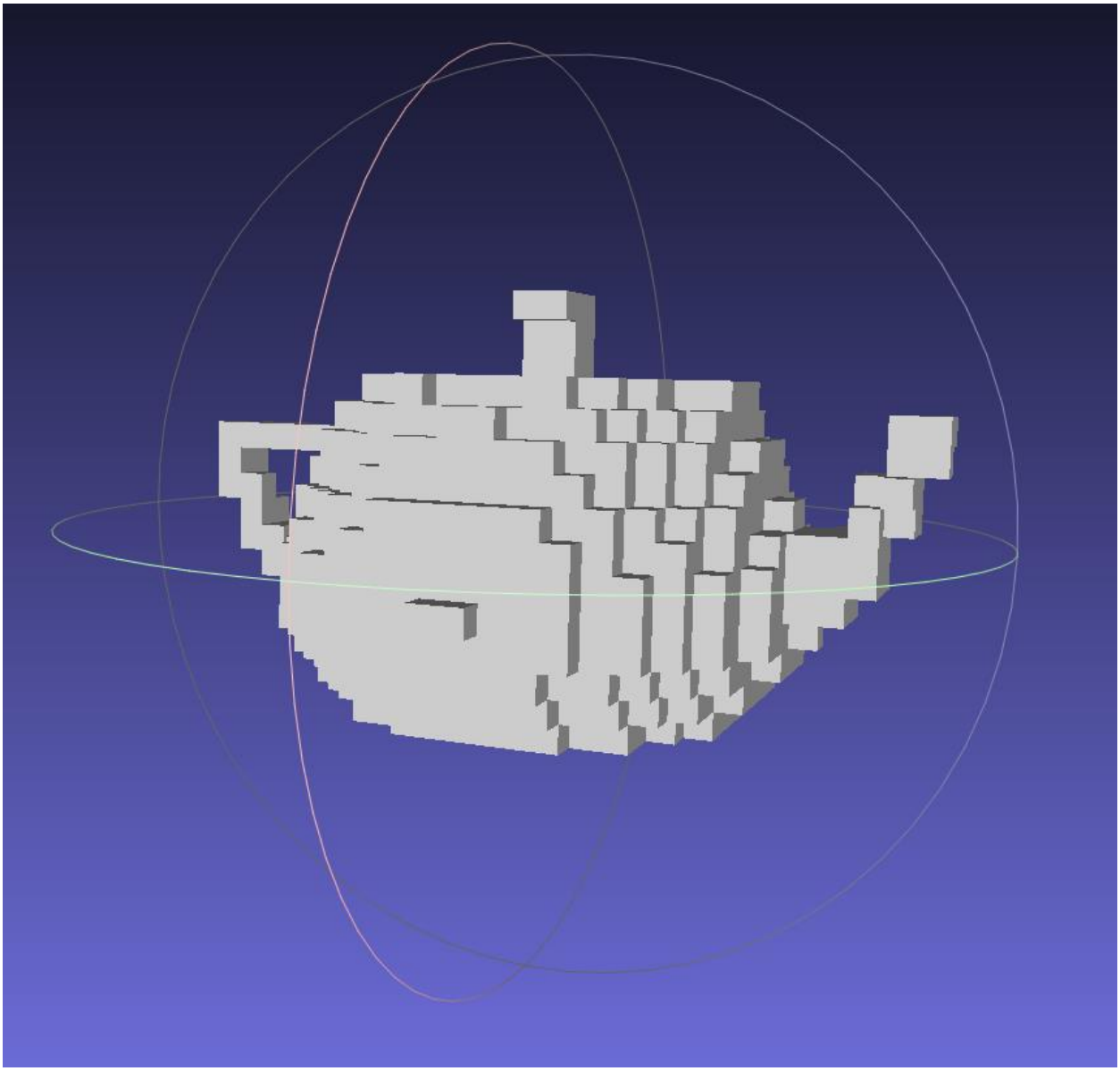
Bunny (32x32x32):



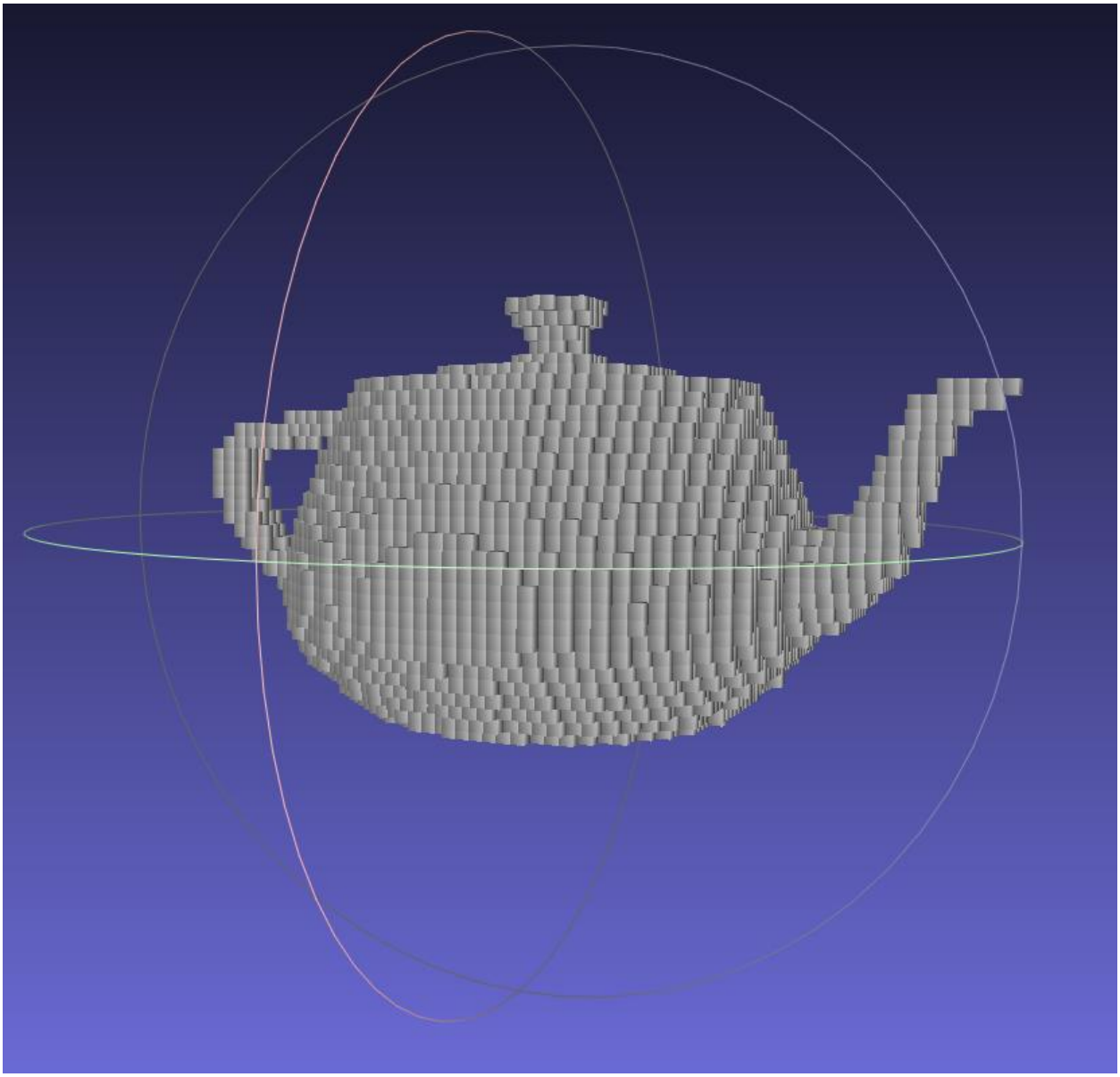
Bunny (64x64x64):



Teapot (32x32x32):



Teapot (64x64x64):



References that I found particularly helpful for completing this assignment

Lecture notes & course webpage: http://people.sutd.edu.sg/~saikit/courses/01.110/

Ray-triangle intersection test: https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/ray-triangle-intersection-geometric-solution

Polar to cartesian transformation: https://blog.demofox.org/2013/10/12/converting-to-and-from-polar-spherical-coordinates-made-easy/

Known problems with my code: none, as far as I know

Extra credit

I implemented the portion to cast a ray in a multiple directions as follows:

```
std::vector<CompFab::Vec3> newDirections;
float theta_r, XTheta, YTheta, phi_r, XPhi, YPhi, X, Y, Z;
int stepSize = 60;
for(int theta=0; theta<360; theta=theta+stepSize){
	for(int phi=0; phi<360; phi=phi+stepSize){
		theta_r = theta * PI / 180.0;
		phi_r = phi * PI / 180.0;
		XTheta = cos(theta_r);
		YTheta = sin(theta_r);
		XPhi = cos(phi_r);
		YPhi = sin(phi_r);
		X = XTheta * XPhi;
		Y = YTheta * XPhi;
		Z = YPhi;
		newDirections.push_back(CompFab::Vec3(X, Y, Z));
	}
}
```

The stepSize variable decides what is the interval (in degrees) between one ray to its nearest neighbour. It is set to 60 by default for faster execution when testing.

The two for loops generates the theta and phi angles needed for casting ray in a sphere (polar coordinates). Within the loops, the angle is converted from degree to radius, and then used to convert

the point to cartesian (X, Y, Z). The radius of the ray is set to 1 by default, as the magnitude of the non-zero directional vector is not affecting ray direction.

```
g_voxelGrid->isInside(i, j, k) = false;
for(int index=0; index<newDirections.size(); index++){
	if (g_voxelGrid->isInside(i, j, k)){
		break;
	}
	else if(numSurfaceIntersections(voxelPos, newDirections[index]) % 2 !=0) {
		g_voxelGrid->isInside(i, j, k) = true;
	}
}
```

This part of the code loops over all the newDirection vectors in the newDirections array, and conducts a ray-triangle intersection test over all directions for each voxel.