Project 3

Natasha Anisimova

.glib file

```
##OpenGL GLIB
Perspective 90
LookAt 0 0 3  0 0 0  0 1 0

Vertex   checkers.vert
Fragment checkers.frag
Program  Checkers                   \
                uUseST <false>                      \
                uSmooth <false>             \
                uSquareColor {1. .5 0.} \
                uAd <.01 .05 .5>  \
                uBd <.01 .05 .5>  \
     uTol <0. 0. 1.>          \
     uNoiseAmp <0. 0. 10.>  \
                uNoiseFreq <0. 1. 10.>  \
     uAlpha <0. 1. 1.>

Color 1  1.  1.
#Torus  1.  .5  60 60
Sphere 1. 60 60
```

.vert file

```
#version 330 compatibility

out vec4  vColor;
out float vLightIntensity;
out vec2  vST;
out vec3 vMCposition;

const vec3 LIGHTPOS = vec3( 0., 0., 10. );

void
main( )
{

	vec3 tnorm = normalize( gl_NormalMatrix * gl_Normal );
	vec3 ECposition = ( gl_ModelViewMatrix * gl_Vertex ).xyz;
	vLightIntensity  = abs( dot( normalize(LIGHTPOS - ECposition), tnorm ) );

	vColor = gl_Color;
	vST = gl_MultiTexCoord0.st;

	vMCposition = gl_Vertex.xyz;
	gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

.frag

```
#version 330 compatibility

in vec4  vColor;
in float vLightIntensity;
in vec2  vST;
in vec3 vMCposition;

uniform bool  uUseST;
uniform bool  uSmooth;
uniform float uSize;
uniform vec4  uSquareColor;
uniform float uAd;
uniform float uBd;
uniform float uTol;
uniform float uNoiseAmp;
uniform float uNoiseFreq;
uniform float uAlpha;
uniform sampler3D Noise3;



const float THREE16    = 3./16.;

float
Pulse( float value, float left, float right, float tol )
{
    float t = (  smoothstep( left-tol, left+tol, value )  -  smoothstep( right-tol, right+tol, value )
);
    return t;
}


void
main( void )
{
    float tryNoiseFreq = uNoiseFreq;
    vec3 stp = uNoiseFreq *vMCposition;
    vec4  nv  = texture( Noise3, stp );
    float n = nv.r + nv.g + nv.b + nv.a;    // 1. -> 3.
    n = n - 2.;                             // -1. -> 1.
    float delta = uNoiseAmp * n;


    float V;
    if( uUseST )
        V = vST.t;
    else
        V = vMCposition.x;
    ///////////////////////////////////////////////////////
    float   Width = 0.10,                       // square width
            Ramp = 0.1,                         // fraction of square used in the ramp
            Height = 0.2,                       // displacement height
            Ad = 0.25,
            Bd = 0.10,
            NoiseAmp = 0.00,
            DispAmp = 0.10;
    float halfSize = uSize/2.;
    float f = fract(  uAd*(V+delta) );

    float TheHeight = 0.;                // how much displacement to apply



    float s = vST.s;
    float t = vST.t;
    float sp = 2. * s;            // good for spheres
    float tp = t;
    //int numins = int( sp / uSize );
    //int numint = int( tp / uSize );

    gl_FragColor = vColor;               // default color

    float up = 2. * s;
    float vp = t;
    up += delta;
    vp += delta;
    float numins = floor( up / uAd ); //Ad
    float numint = floor( vp / uBd ); //Bd

    //point PP = point P; //point "shader" P;
    float magnitude = 0.;
    float size = 1.;
    float i;
```

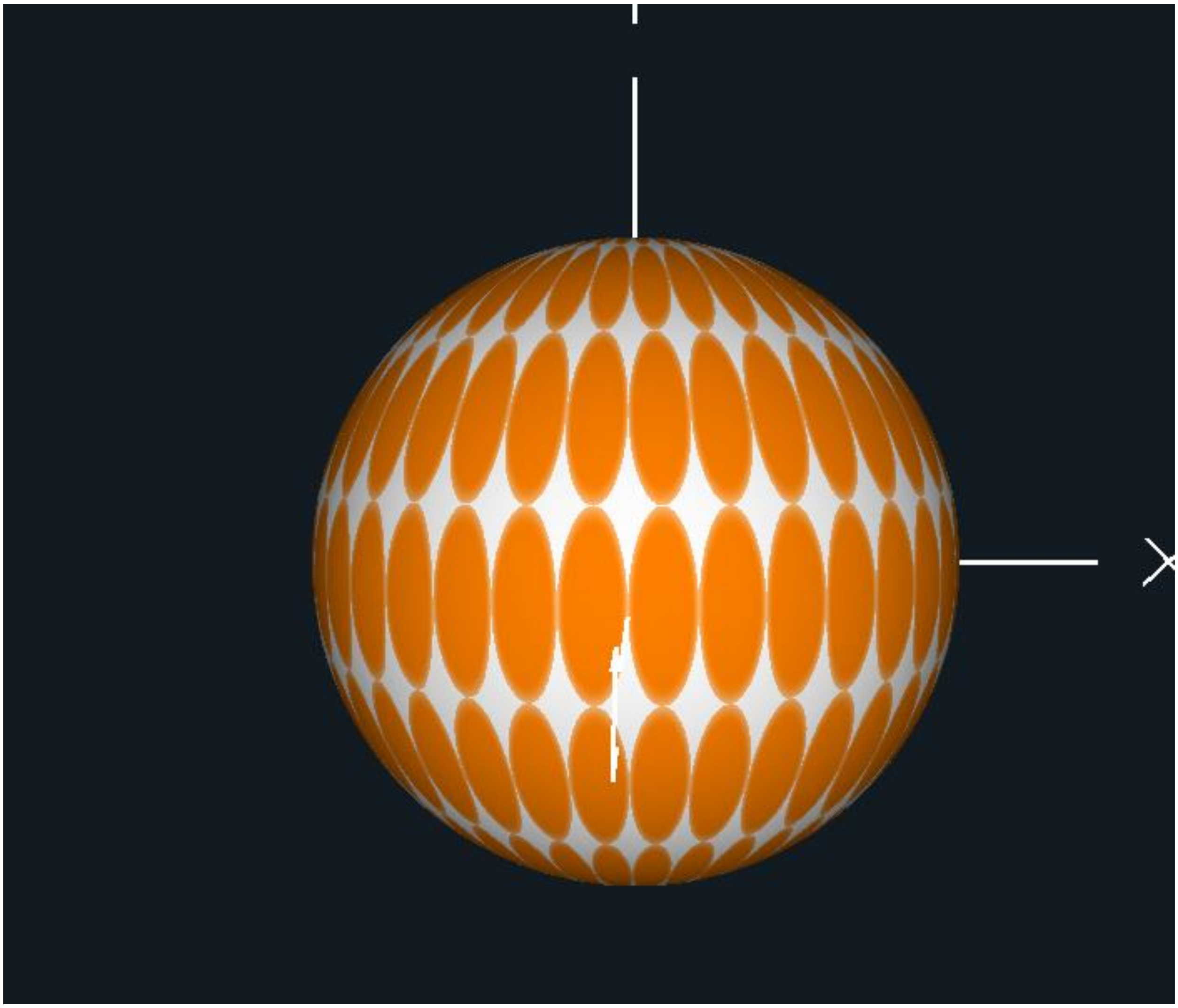
```
    float uc = uAd *numins + (uAd/2);
    float vc = uBd *numint+ (uBd/2);
    float r = Width/2.;
    float Ar = uAd/2.;
    float Br = uBd/2.;
    float du = up - uc;
float dv = vp - vc;
float oldrad = sqrt( du*du + dv*dv );
    float newrad = magnitude + oldrad;
    float factor = newrad/oldrad;
    du *= factor;
    dv *= factor;

    float d = pow((du)/Ar, 2.) + pow((dv)/Br, 2.);

    if (d <= 1.)
    {
        if( uSmooth )
        {
            //float t = 1. - d;//this is if you want the inverse smooth shading
            float t= smoothstep( 1.-uTol, 1.+uTol, d );
            gl_FragColor = mix( uSquareColor, vColor, t );
            TheHeight = t*Height;
            V = vST.t;
        }
        else
        {
            gl_FragColor = uSquareColor;
            V = vMCposition.x;
        }
    }
    if(gl_FragColor == vColor){
        gl_FragColor.w = uAlpha;
        if (uAlpha == 0){
            discard;
        }
    }


    if( f >= d - uTol )
    {
        t = smoothstep( 1.-uTol, 1.+uTol, d  );
        gl_FragColor = mix( uSquareColor, vColor, t );
    }
    gl_FragColor.rgb *= vLightIntensity;// apply lighting model
}
```

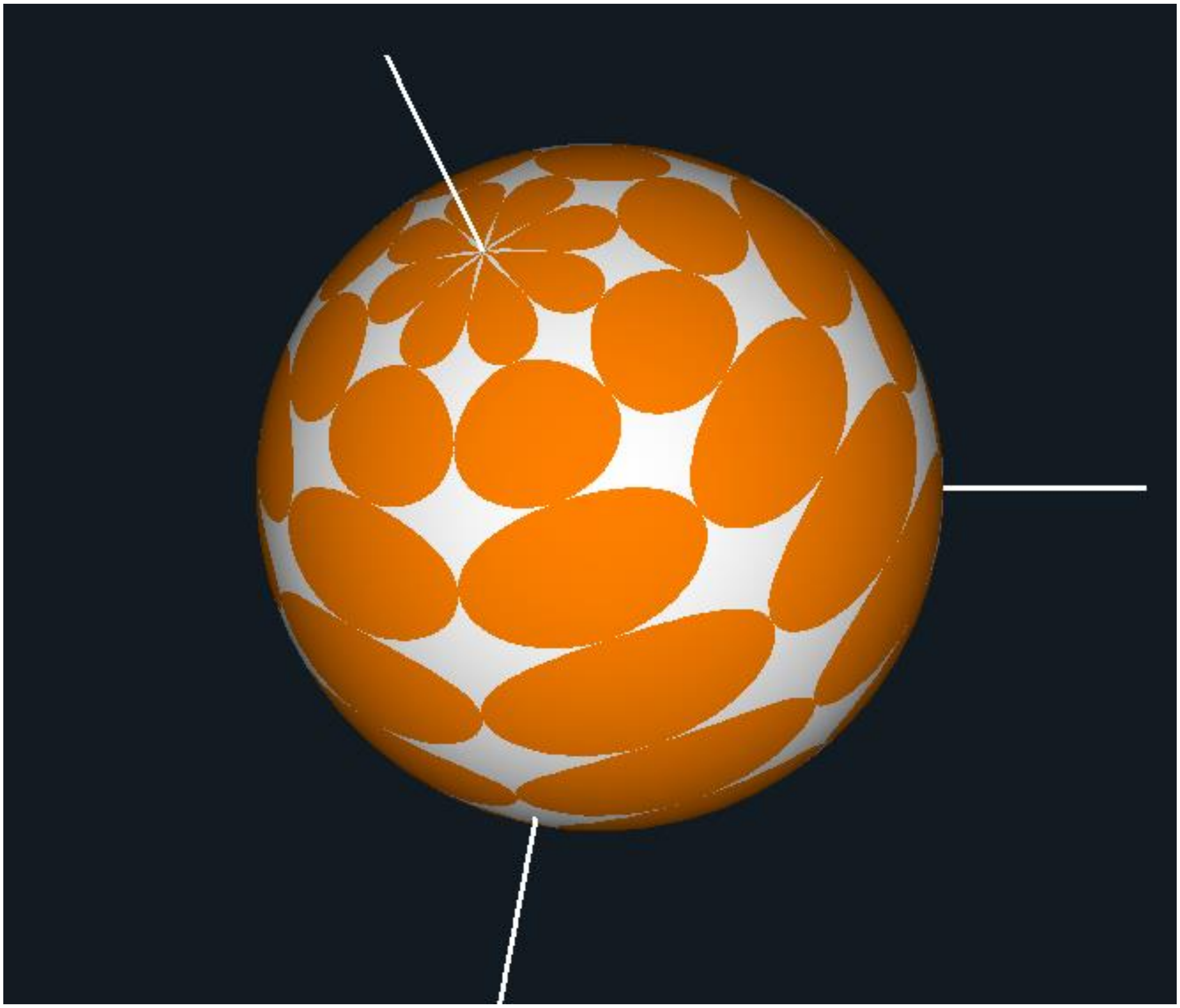
Explanation

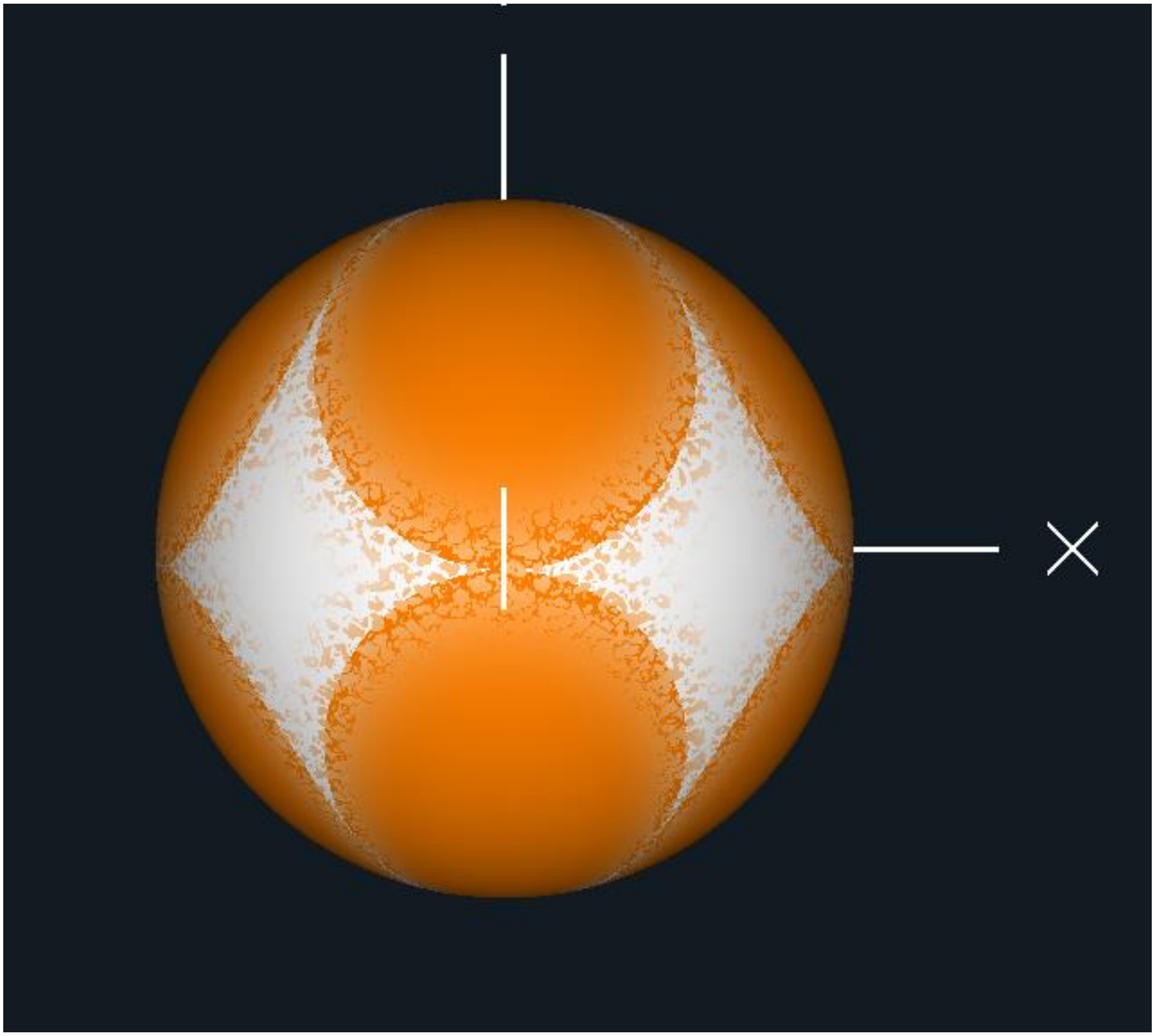
What I did for this project was simply take the checkers.glib, checkers.frag, and checkers.vert files and convert them to be ellipses first. After the ellipses were successfully created with the equation that I had used in previous projects, I attempted to do the smooth step (interestingly enough it looks strange when you add on the discard based upon color). I even tried doing the inverse of the smoothstep so that the inside of the ellipses were white which gave it a cool effect.

What came next threw me for a few loops. When trying to play around with noise I first could only get it to mess around with what seemed to be the lighting, and then the edges of the smoothstep ellipses. It was all based of how I was trying to add the noise, so it all can be replicated again if needed. I also made the range for the variables for noise (NoiseFreq and NoiseAmp) really large just to see what could happen. At one point I was getting what looked like Jupiter and something that looked like it could be a swirled candy. What I noticed that was different with using glman, or at least one of things, was that we had a sampler3D Noise3 that we could import. Overall I really enjoyed this project. Finding all the ways that noise could be applied to a shape to create patterns, or not, was visually engaging.

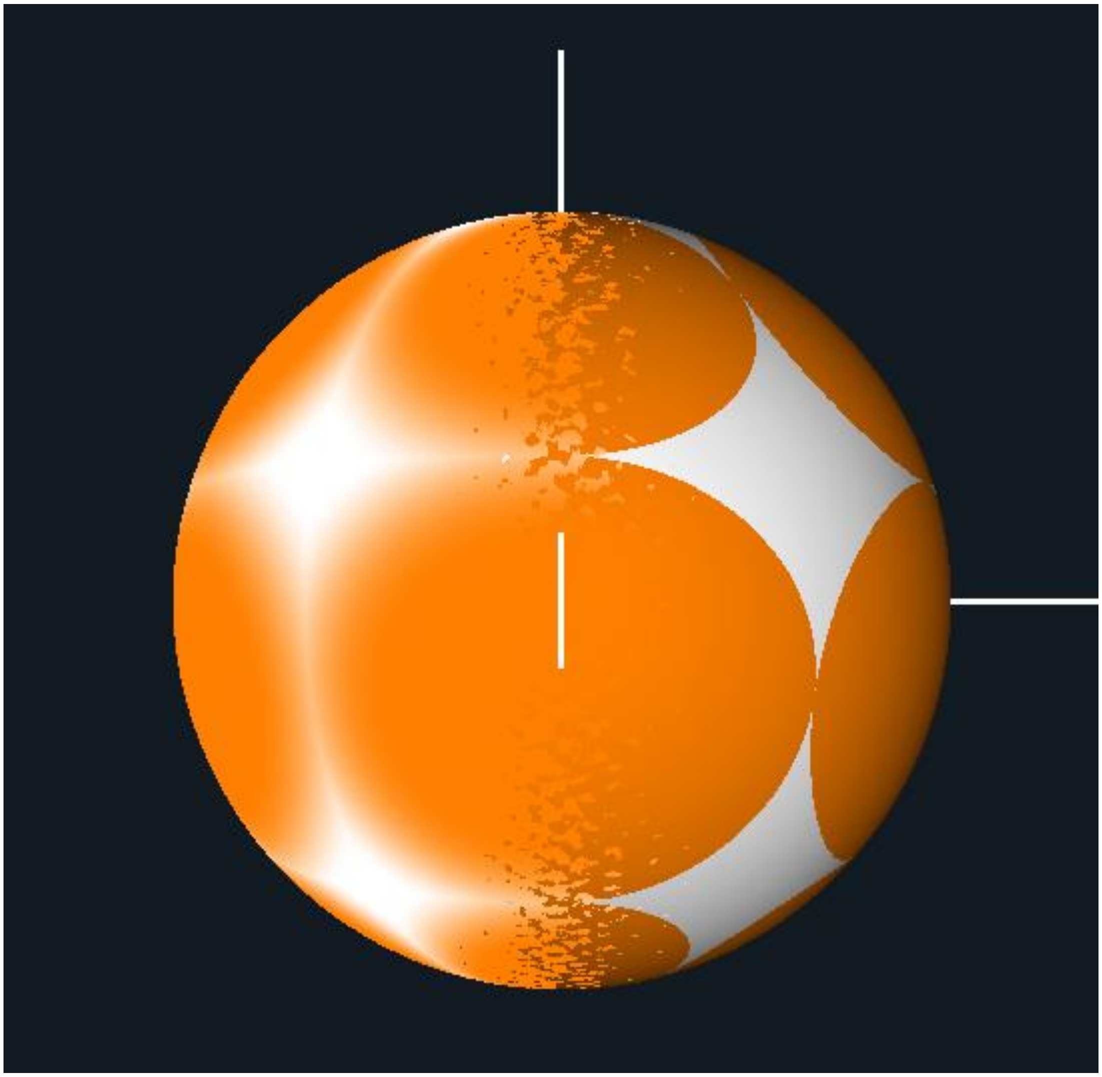
After the noise was working correctly for the project, it was fairly simply to apply the discard to just the white parts of the object (image found below). All that needed to be set was uAlpha. I also have a weird variable called uUSSET that will turn off certain features for half of the sphere. I decided to keep it so that the person who looks at the code could play around with it.

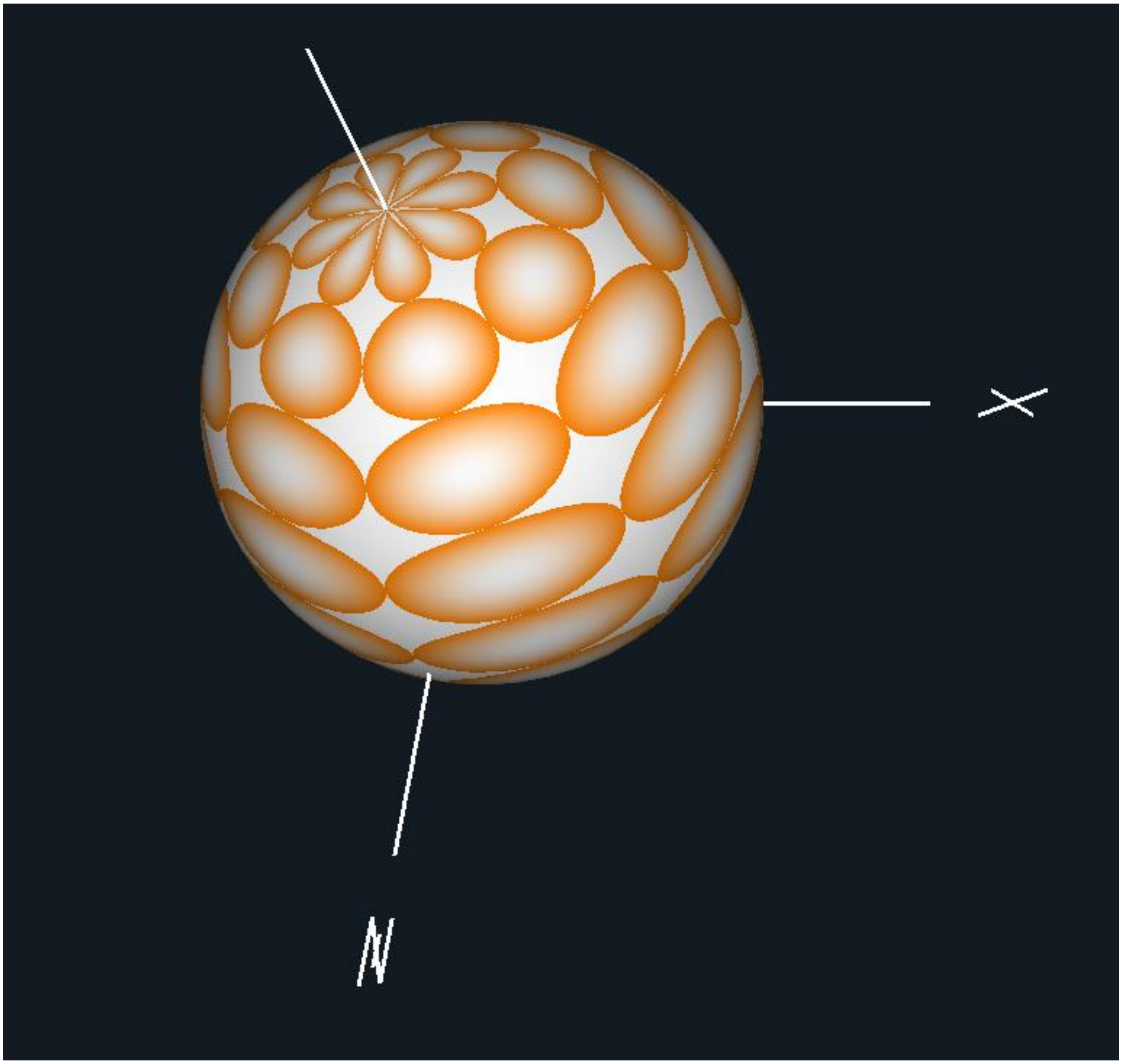

X

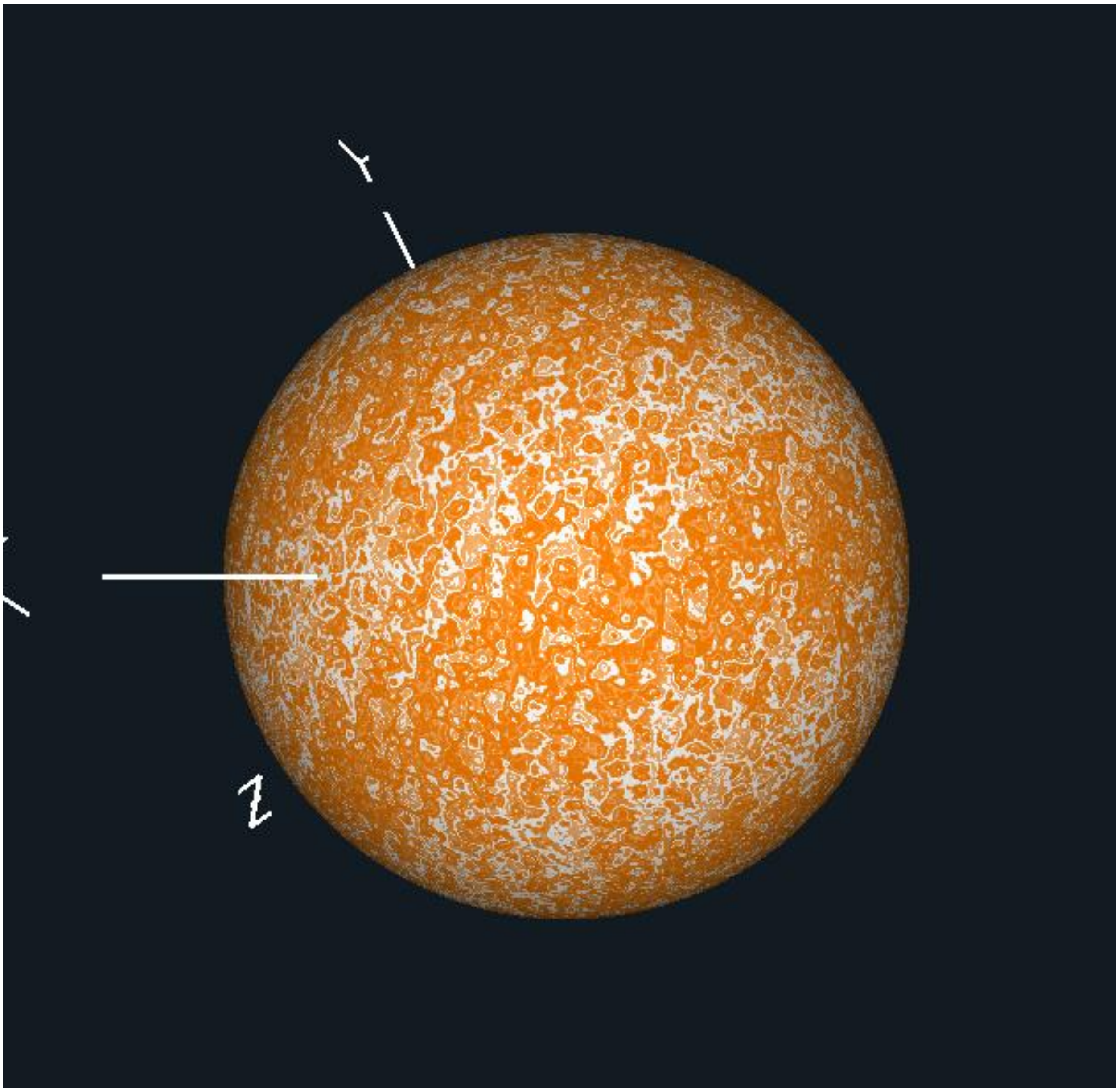




X




X
N


Y
Z