CS 557 -- Winter Quarter 2016

Project #4 Report

Displacement Mapping, Lighting, and Bump Mapping

Name: Kai Shi

ID: 932-504-512

In project4, I used displacement mapping to turn a simple shape into a more interesting one, re-compute its normals, light it, and bump-map it.

(1) Code

glib:

```
##OpenGL GLIB
Perspective 70
LookAt 0 0 3  0 0 0  0 1 0
Vertex kaishi.vert
Fragment        kaishi.frag
Program    kaishi                                       \
                    uKa <0. 0.1 1.0>                    \
                    uKd <0. 0.6 1.0>                    \
                    uKs <0. 0.3 1.0>                    \
                    uShininess <3. 10. 1000.>           \
                    uLightX <-20. 5. 20.>               \
                    uLightY <-20. 10. 20.>              \
                    uLightZ <-20. 20. 20.>              \
                    uColor {1. .7 0. 1.}                \
                    uSpecularColor {1. 1. 1. 1.}   \
                    uA <-2.0 0.00 2.0>                  \
                    uB <0.0 5.0 20.0>                   \
                    uC <0.0 5.0 20.0>                   \
                    uNoiseAmp <0. 0. 5.>                \
      uNoiseFreq <0.1 1. 20.>
Translate -1.  1.
QuadXY  -0.2  1.  50 50
```

vert:

```
#version 330 compatibility
uniform float uLightX, uLightY, uLightZ;
out vec3 vNs;
out vec3 vLs;
out vec3 vEs;
out vec3 vMC;
uniform float uA, uB, uC;
```

```
vec3 eyeLightPosition = vec3( uLightX, uLightY, uLightZ );

void
main( )
{
  vMC = gl_Vertex.xyz;
  vec4 new_vertex = gl_Vertex;
  new_vertex.z = uA * cos(uB*gl_Vertex.x) * cos(uC*gl_Vertex.y);
  vMC.z = new_vertex.z;
  float dzdx = -uA * uB * sin(uB*new_vertex.x) * cos(uC*new_vertex.y);
  float dzdy = -uA * uC * cos(uB*new_vertex.x) * sin(uC*new_vertex.y);
  vec3 Tx = vec3(1.,0.,dzdx);
  vec3 Ty = vec3(0.,1.,dzdy);
  vec3 new_normal = normalize(cross(Tx,Ty));
  vec4 ECposition = gl_ModelViewMatrix * new_vertex;
  vNs = normalize( gl_NormalMatrix * new_normal );  // surface normal vector
  vLs = eyeLightPosition - ECposition.xyz;    // vector from the point
  vEs = vec3( 0., 0., 0. ) - ECposition.xyz;    // vector from the point
 gl_Position = gl_ModelViewProjectionMatrix * new_vertex;
}
```

frag:

```
#version 330 compatibility
uniform float uKa, uKd, uKs;
uniform vec4  uColor;
uniform vec4  uSpecularColor;
uniform float uShininess;
uniform float uNoiseAmp;
uniform float uNoiseFreq;
uniform sampler3D Noise3;

in vec3 vNs;
in vec3 vLs;
in vec3 vEs;
in vec3 vMC;

vec3
RotateNormal( float angx, float angy, vec3 n )
{
        float cx = cos( angx );
```

```
        float sx = sin( angx );
        float cy = cos( angy );
        float sy = sin( angy );

        // rotate about x:
        float yp =  n.y*cx - n.z*sx;    // y'
        n.z      =  n.y*sx + n.z*cx;    // z'
        n.y      =  yp;
        // n.x      =  n.x;
        // rotate about y:
        float xp =  n.x*cy + n.z*sy;    // x'
        n.z      = -n.x*sy + n.z*cy;    // z'
        n.x      =  xp;
        // n.y      =  n.y;
        return normalize( n );
}
void
main( )
{
  vec3 Normal;
  vec3 Light;
  vec3 Eye;

  vec4 nvx = uNoiseAmp * texture3D( Noise3, uNoiseFreq*vMC );
  float angx = nvx.r + nvx.g + nvx.b + nvx.a;     //  1. -> 3.
  angx = angx - 2.;                               // -1. -> 1.
  angx *= uNoiseAmp;

  vec4 nvy = uNoiseAmp * texture3D( Noise3, uNoiseFreq*vec3(vMC.xy,vMC.z+0.5) );
  float angy = nvy.r + nvy.g + nvy.b + nvy.a;     //  1. -> 3.
  angy = angy - 2.;                               // -1. -> 1.
  angy *= uNoiseAmp;

  Normal = normalize(vNs);
  Light =  normalize(vLs);
  Eye =    normalize(vEs);

  Normal = RotateNormal(angx, angy, Normal);

  vec4 ambient = uKa * uColor;
```

```
  float d = max( dot(Normal,Light), 0. );
  vec4 diffuse = uKd * d * uColor;

  float s = 0.;
  if( dot(Normal,Light) > 0. )    // only do specular if the light can see the point
  {
    vec3 ref = normalize( 2. * Normal * dot(Normal,Light) - Light );
    s = pow( max( dot(Eye,ref),0. ), uShininess);
  }
  vec4 specular = uKs * s * uSpecularColor;
  gl_FragColor = vec4( ambient.rgb + diffuse.rgb + specular.rgb, 1. );
}
```

(2) What I did and Reasons

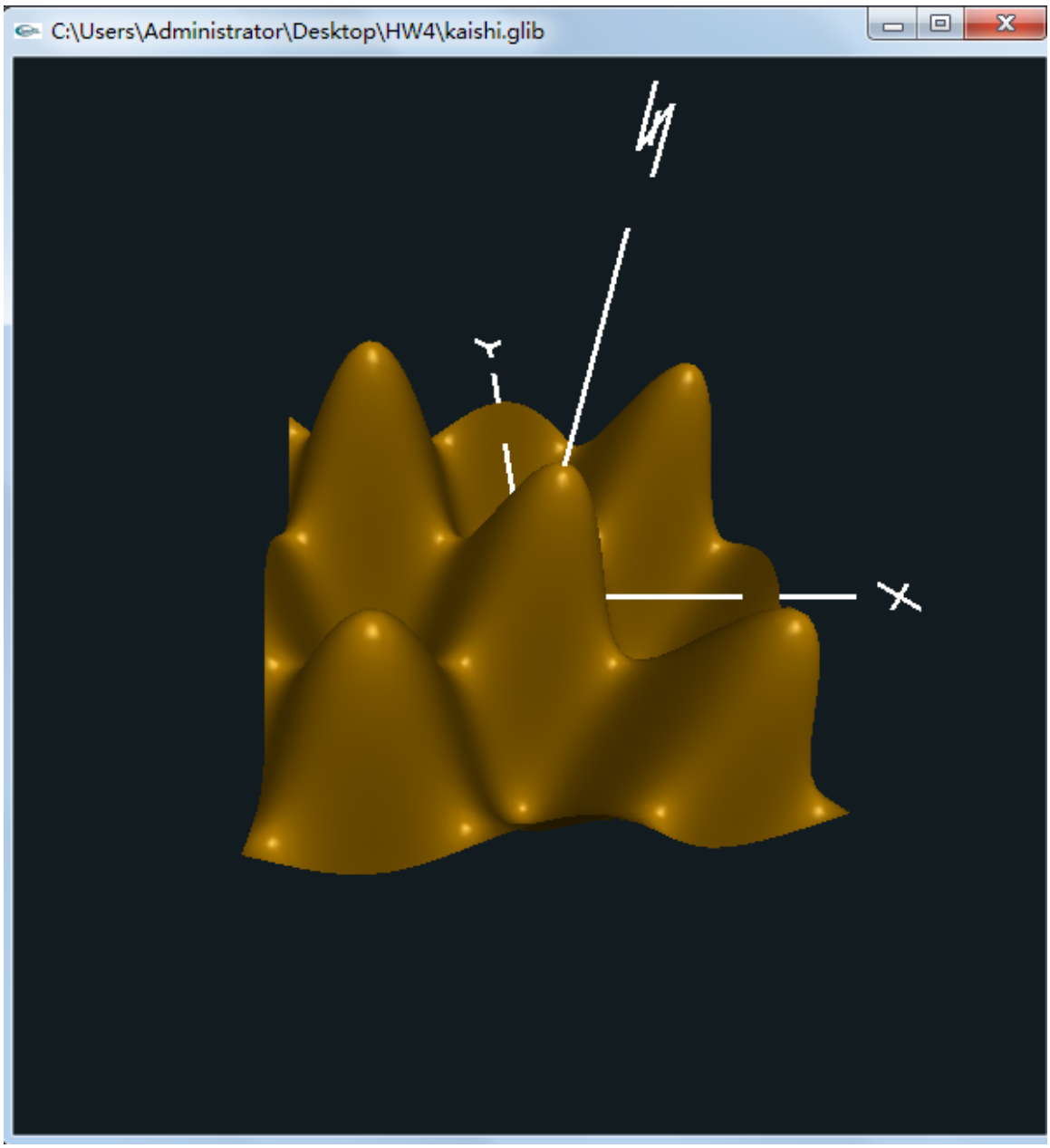
Because Z is 0 at the beginning, so I need to use Z = A * cos(Bx) * cos(Cy) to rise Z and make it like hills. After this step, I can get a dark object and I can make it like waves by changing uA, uB, uC.

Next step is adding lighting. I use the per-fragment lighting. In the vertex shader, I computed the new normal about the vector which is vertical with Tx, Ty (). The vNs = normalize(gl_NormalMatrix * new normal). Transmitting it to the fragment shader. Using it in the per-frag lighting. After this step, I finished the turning waves and lighting parts.

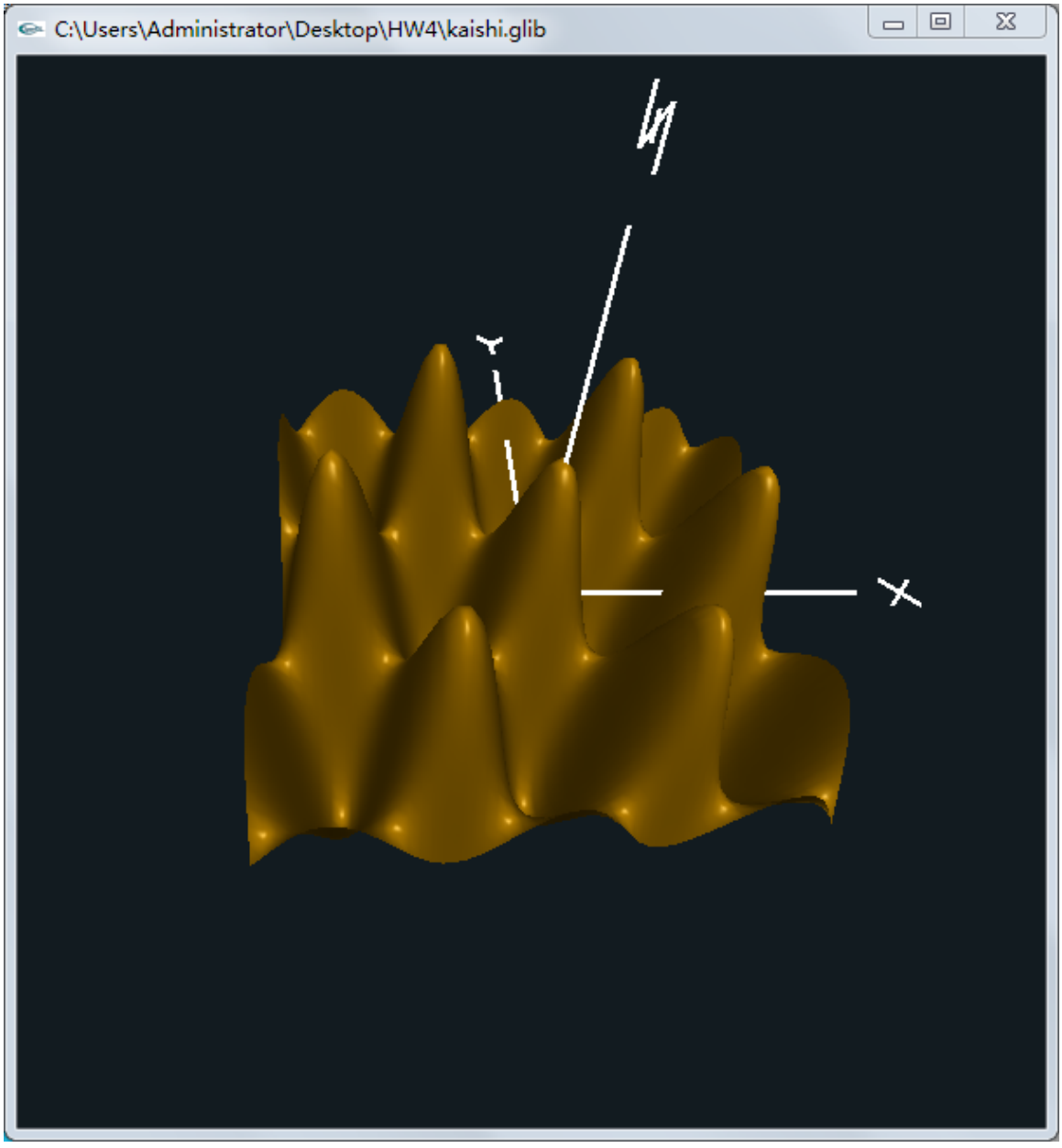
Last step is bump-mapping. Adding noise. I write this part in the fragment shader. I used vec4 nvx = uNoiseAmp * texture3D(Noise3, uNoiseFreq*vMC); vec4 nvy = uNoiseAmp * texture3D(Noise3, uNoiseFreq*vec3(vMC.xy,vMC.z+0.5)); to create NoiseAmp and NoiseFreq. In this step, using float angx = nvx.r + nvx.g + nvx.b + nvx.a and angx = angx – 2 to limit the range between -1 and 1. Then, angx *= uNoiseAmp. The same does angy. Then, using RotateNormal function among vNs, angx and angy.

(3) Results

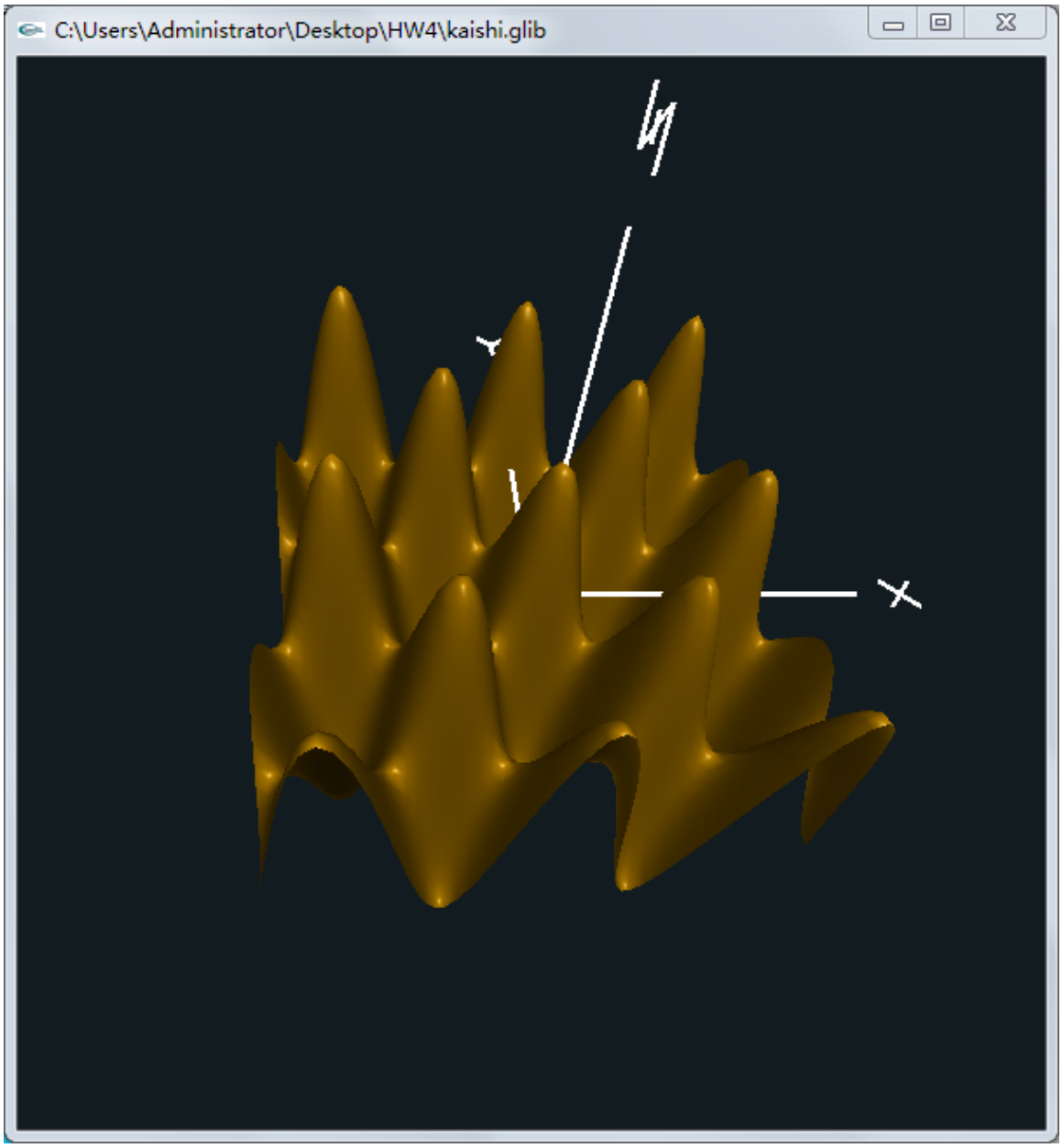
Changing uA:



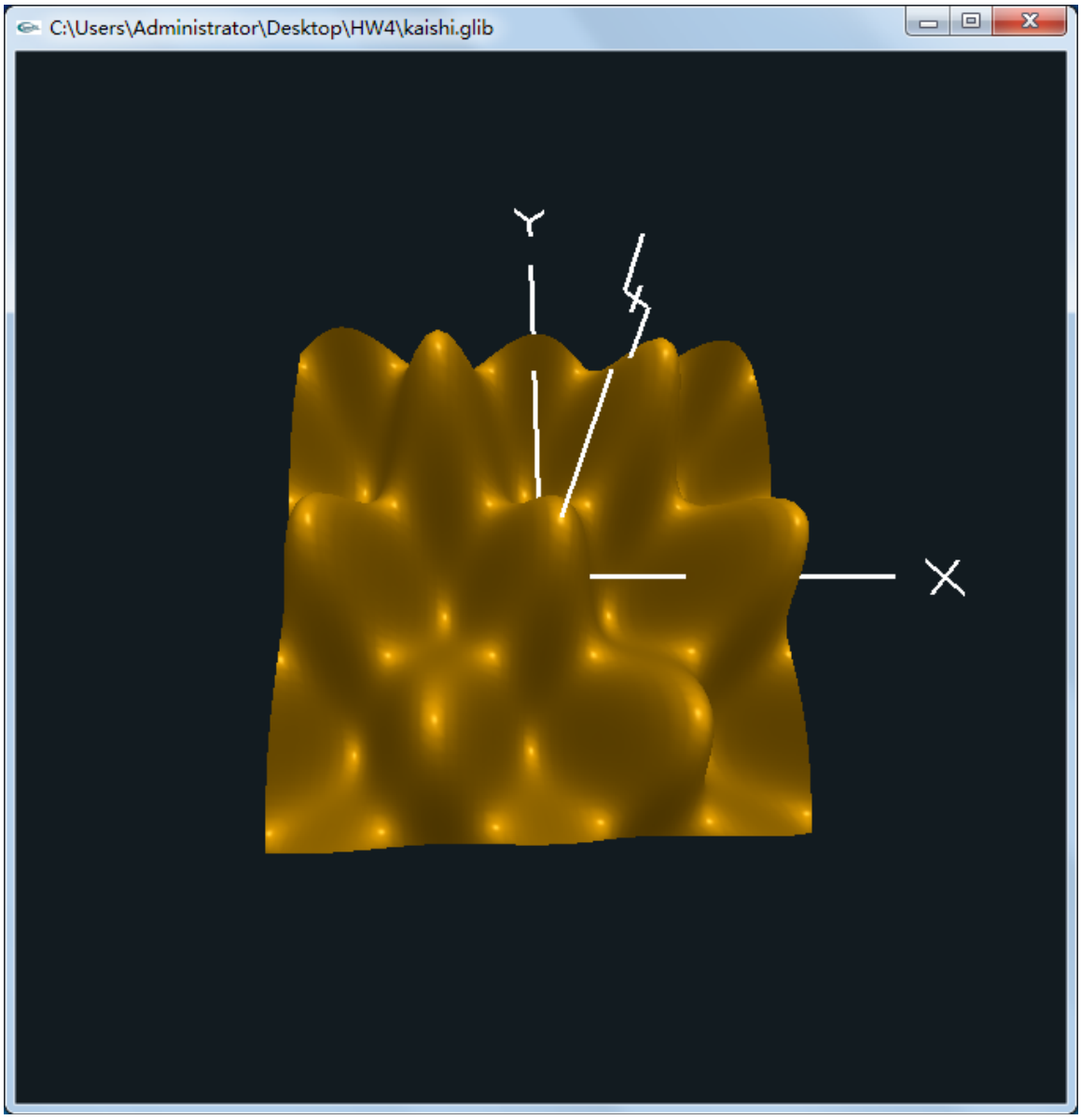
Changing uB:



Changing uC:



Changing lighting position and uKa, uKd and uKs:



Bump-mapping: