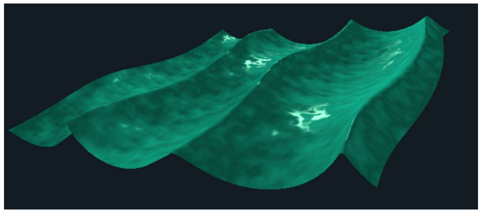
Animating Wave Motion using Gerstner Waves

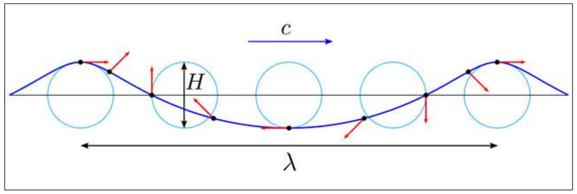
Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu



How Do Waves Work?

First of all, the water in waves doesn’t “flow”. It moves in a circular pattern. The equation for this is called a Trochoidal wave, or a Gerstner wave, named after mathematician Franz Josef Gerstner who discovered this in 1802.

Click on the Wikipedia link below for more information. It’s an interesting read.



https://en.wikipedia.org/wiki/Trochoidal_wave

If you scroll down in the Wikipedia article, you will see a section called **In Computer Graphics**. I adapted the following equations and code from that section. I am assuming deep water so that the hyperbolic tangent term drops out. Feel free to put it back. I also changed the wave density components to an angular direction (γ) instead.

Gerstner Wave Equations

$$\text{Horizontal Motion} \rightarrow x' = x - \sum_{m=0}^{M-1} A_m cos\gamma_m sin\theta_m$$

$$\text{Vertical Motion} \rightarrow y' = \sum_{m=0}^{M-1} A_m cos\theta_m$$

$$\text{Horizontal Motion} \rightarrow z' = z - \sum_{m=0}^{M-1} A_m sin\gamma_m sin\theta_m$$

$(x, y, z) \quad = original\ vertex\ coordinates$

$(x', y', z') = displaced\ vertex\ coordinates$

$$\theta_m = k_m cos\gamma_m x + k_m sin\gamma_m y - \omega_m t - \emptyset_m$$

$A_m = Amplitude$ $\qquad$ $\omega_m = \sqrt{gk_m}$

$\gamma_m = Wave\ propagation\ angle$ $\qquad$ $t = time$

$k_m = Wave\ density$ $\qquad$ $\emptyset_m = Wave\ phase\ shift$

gerstner.glib

```
##OpenGL GLIB
Perspective 70
LookAt 0 0 7  0 0 0  0 1 0
Timer 60

Vertex   gerstner.vert
Fragment gerstner.frag
Program  Gerstner                                    \
                uTimeScale <1. 2. 100.>              \
\
                uAm0 <0. .2 1.>                      \
                uKm0 <0.1 1. 5.>                     \
                uGamma0 <-1.57080 0. 1.57080>        \
\
                uAm1 <0. 0. 1.>                      \
                uKm1 <0.1 2. 5.>                     \
                uPhiM1 <0. 0. 6.28>                  \
                uGamma1 <-1.57080 0. 1.57080>        \
\
                uLightX <-20. 0. 20.>                \
                uLightY <1. 10. 20.>                 \
                uLightZ <-20. -20. 20.>              \
                uKa <0. .1 1.>                       \
                uKd <0. .6 1.>                       \
                uKs <0. .3 1.>                       \
                uShininess < 1. 2. 200.>             \
                uColor {.1 1. .8 1.}                 \
                uNoiseAmp <0. 0. 1.>                 \
                uNoiseFreq <.1 .1 2.>                \
\

QuadXZ  -0.2  3.  300  300
```

gerstner.vert, I

```
#version 330 compatibility

uniform float       uTimeScale;
//uniform float     uG;
//uniform float     uH;

uniform float       uAm0;
uniform float       uKm0;
uniform float       uGamma0;

uniform float       uAm1;
uniform float       uKm1;
uniform float       uPhiM1;
uniform float       uGamma1;

uniform float       Timer;

uniform float       uLightX, uLightY, uLightZ;
vec3 eyeLightPosition = vec3(  uLightX,  uLightY,  uLightZ );

out vec3                        vMC;
out vec3                        vEs;
out vec3                        vLs;
out vec3                        vNs;

const float PI = 3.14159265;
const float G  = 1.;

void
main( )
{
            float newx = gl_Vertex.x;
            float newy = 0.;
            float newz = gl_Vertex.z;

            float dxda = 1.;
            float dyda = 0.;
            float dzda = 0.;

            float dxdb = 0.;
            float dydb = 0.;
            float dzdb = 1.;
```

gerstner.vert, II

```
            // m = 0
            {
                        float phiM0 = 0.;  // m=0 is the phase baseline
                        float wm0 = sqrt( G*uKm0 );
                        float thetam = gl_Vertex.x*uKm0*cos(uGamma0)+ gl_Vertex.z*uKm0*sin(uGamma0) - wm0*Timer*uTimeScale - phiM0;
                        newx -= uAm0*cos(uGamma0)*sin(thetam);
                        newy += uAm0 * cos(thetam);
                        newz -= uAm0*sin(uGamma0)*sin(thetam);

                        float dthetamda = uKm0*cos(uGamma0);
                        float dthetamdb = uKm0*sin(uGamma0);
                        dxda -= uAm0*cos(uGamma0)*cos(thetam)*dthetamda;
                        dyda -= uAm0*sin(thetam)*dthetamda;
                        dzda -= uAm0*sin(uGamma0)*cos(thetam)*dthetamda;
                        dxdb -= uAm0*cos(uGamma0)*cos(thetam)*dthetamdb;
                        dydb -= uAm0*sin(thetam)*dthetamdb;
                        dzdb -= uAm0*sin(uGamma0)*cos(thetam)*dthetamdb;
            }

            // m = 1
            {
                        float wm1 = sqrt( G*uKm1 );
                        float thetam = gl_Vertex.x*uKm1*cos(uGamma1)+ gl_Vertex.z*uKm1*sin(uGamma1) - wm1*Timer*uTimeScale - uPhiM1;
                        newx -= uAm1*cos(uGamma1)*sin(thetam);
                        newy += uAm1 * cos(thetam);
                        newz -= uAm1*sin(uGamma1)*sin(thetam);

                        float dthetamda = uKm1*cos(uGamma1);
                        float dthetamdb = uKm1*sin(uGamma1);
                        dxda -= uAm1*cos(uGamma1)*cos(thetam)*dthetamda;
                        dyda -= uAm1*sin(thetam)*dthetamda;
                        dzda -= uAm1*sin(uGamma1)*cos(thetam)*dthetamda;
                        dxdb -= uAm1*cos(uGamma1)*cos(thetam)*dthetamdb;
                        dydb -= uAm1*sin(thetam)*dthetamdb;
                        dzdb -= uAm1*sin(uGamma1)*cos(thetam)*dthetamdb;
            }
```

gerstner.vert, III

```
            vec3 newVertex = vec3( newx, newy, newz );
            vMC = newVertex;

            vec3 ta = vec3( dxda, dyda, dzda );
            vec3 tb = vec3( dxdb, dydb, dzdb );
            vNs = normalize( gl_NormalMatrix*cross( tb, ta ) );
                                                                  // surface normal vector

            vec4 ECposition = gl_ModelViewMatrix * vec4( newVertex, 1. );
            vLs = normalize( eyeLightPosition - ECposition.xyz );  // vector from the point
                                                                  // to the light position
            vEs = normalize( vec3( 0., 0., 0. ) - ECposition.xyz );  // vector from the point
                                                                  // to the eye position

            gl_Position = gl_ModelViewProjectionMatrix * vec4( newVertex, 1.);
}
```

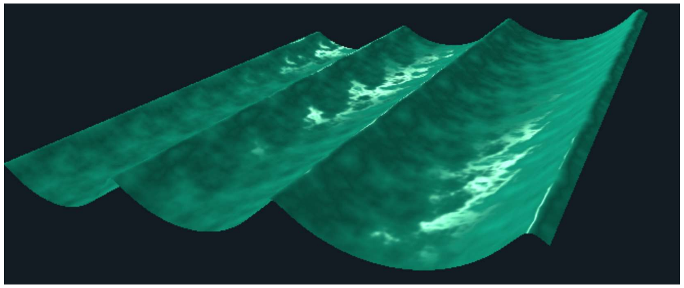
gerstner.frag, I

```
#version 330 compatibility

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

uniform float                   uKa, uKd, uKs;
uniform vec4                    uColor;
uniform float                   uShininess;
uniform sampler3D               Noise3;
uniform float                   uNoiseAmp;
uniform float                   uNoiseFreq;

const vec4 WHITE = { 1., 1., .8, 1. };

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 );
}
```

gerstner.frag, II

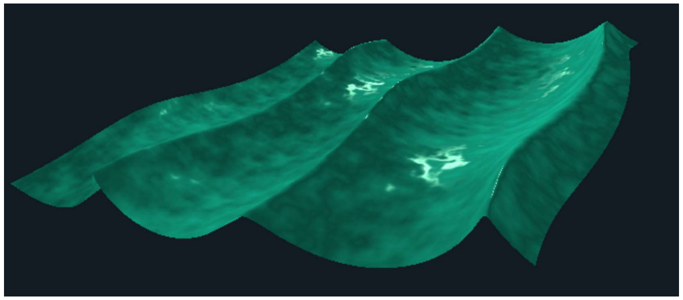
```
void
main( )
{
                vec4 nvx = texture3D( Noise3, uNoiseFreq*vMC );
                vec4 nvy = texture3D( Noise3, uNoiseFreq*vec3(vMC.xy,vMC.z+0.5) );

                float angx = nvx.r + nvx.g + nvx.b + nvx.a;          //  1. -> 3.
                angx = angx - 2.;
                // -1. -> 1.
                angx *= uNoiseAmp;

                float angy = nvy.r + nvy.g + nvy.b + nvy.a;          //  1. -> 3.
                angy = angy - 2.;
                // -1. -> 1.
                angy *= uNoiseAmp;

                vec3 normal = normalize( vNs );
                vec3 light  = normalize( vLs );
                vec3 eye    = normalize( vEs );

                normal = RotateNormal( angx, angy, normal );

                vec4 ambient = uKa * uColor;

                float d = max( dot(normal,light), 0. );
                d = abs( dot(normal,light));
                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 * WHITE;
                gl_FragColor = vec4( ambient.rgb + diffuse.rgb + specular.rgb, 1. );
}
```

Example



m = 0

Example



m = 0, 1