

电子科技大学信息与软件工程学院

# 实 验 指 导 书

(实验) 课程名称 图形与动画 II

电子科技大学教务处制表

# 目 录

实验五 水面模拟 .....	2
一、实验目的.....	2
二、实验原理.....	2
三、实验内容.....	18
四、实验步骤.....	19

# 实验五 水面模拟

## 一、实验目的

1. 掌握实现水面模拟的基本思路，特别是水面高度函数；
2. 实现一个简单的水面模拟（单一振源）；
3. 实现较复杂的水面模拟（多振源）。

## 二、实验原理

### （一）水面模拟原理

#### 1. 水面建模

将需要模拟的水面划分为若干个均匀网格，再对每个网格 $(x,y)$ 坐标求高度函数  $H(x,y,t)$ ：

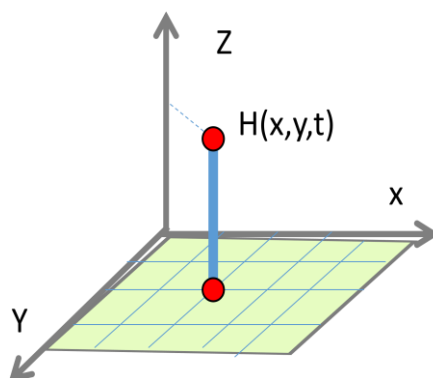


图 1

#### 2. 水面高度函数

二维平面上某点 $(x,y)$ 处振幅大小随时间的变化关系 $H(x,y,t)$ ：

$$H(x,y,t) = A \times \sin(D \cdot (x,y) \times w + t \times \varphi)$$

- 其中  $w = \frac{2\pi}{L}$ ,  $\varphi = S \times \frac{2\pi}{L}$ ;
- 波长 (Wavelength -  $L$ ) : 连续两个波峰之间的距离;
- 振幅 (Amplitude -  $A$ ) : 从平面到波峰的距离;
- 波传播速度 (Speed -  $S$ ) : 波峰每秒钟移动距离;
- 波浪传播方向 (Direction -  $D$ ) : 垂直于波阵面的向量;

法向量:

$$N(x, y) = \left( -\frac{\partial}{\partial x}(H(x, y, t)), -\frac{\partial}{\partial y}(H(x, y, t)), 1 \right)$$

$$\frac{\partial}{\partial x}(H(x, y, t)) = w \times D \cdot x \times A \times \cos(D \cdot (x, y) \times w + t \times \varphi)$$

$$\frac{\partial}{\partial y}(H(x, y, t)) = w \times D \cdot y \times A \times \cos(D \cdot (x, y) \times w + t \times \varphi)$$

其中：

$-D \cdot x$ 为向量 $D$ 中 $x$ 分量；

$-D \cdot y$ 为向量 $D$ 中 $y$ 分量；

## (二) 简单水面模拟 - 一个振源

本节我们将实现一个简单的水面模拟。如图 2 所示，该水面模型只有一个振源，位于原点。

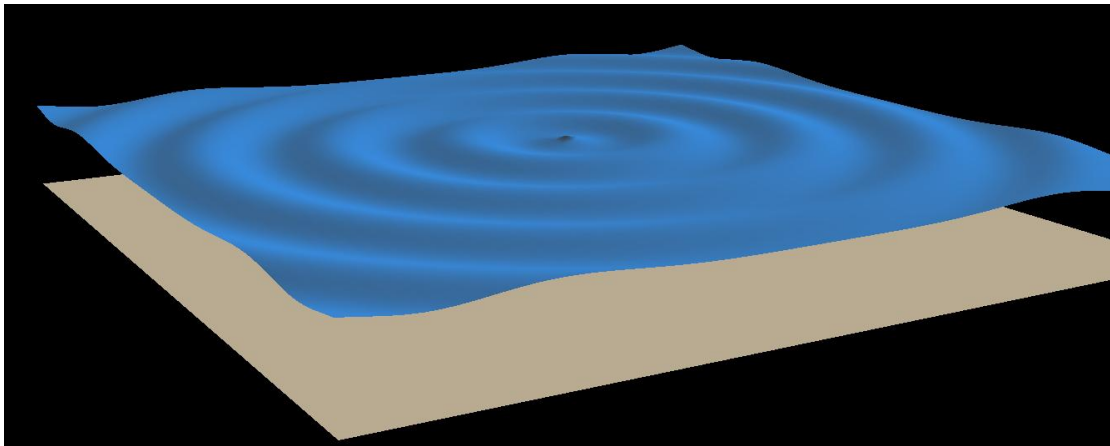


图 2

### 1. 水面高度函数

#### 1) 振源参数

通过如下全局变量设置振源参数：

```
float amplitude = 0.01;
float wavelength = 0.3;
float speed = -0.2;
float center[1][2] = {0.0, 0.0};
```

图 3

振源位于原点，幅度 0.01，波长 0.3，传播速度 0.2，负号控制其传播方向。

#### 2) 水面高度

对比水面高度公式，写出水面高度函数，如图 4 所示。注意，函数 **waveHeight** 计算位于位置  $(x, y)$  处，时间  $t$  水面高度。其中函数 **dot** 计算了网格点  $(x, y)$  距离振源  $(0, 0)$  的距离：

```

float dot(float x, float y){
    float cx = x - center[0][0];
    float cy = y - center[0][1];
    return sqrt(cx * cx + cy * cy);
}

float waveHeight(float x, float y, float time){
    float frequency = 2*PI/wavelength;
    float phase = speed* frequency;
    float theta = dot(x,y);

    return amplitude * sin(theta * frequency + time * phase);
}

```

图 4

### 3) 法线计算

函数 **dWavedx** 和 **dWavedy** 分别计算了函数  $H(x,y,t)$  在网格点  $(x,y)$  处  $x$ ,  $y$  方向上的偏导数:

```

float dWavedx(float x, float y, float time){
    float frequency = 2 * PI / wavelength;
    float phase = speed * frequency;
    float theta = dot(x, y);
    float A = amplitude * x * frequency / theta;
    return A * cos(theta * frequency + time * phase);
}

float dWavedy(float x, float y, float time){
    float frequency = 2 * PI / wavelength;
    float phase = speed * frequency;
    float theta = dot(x, y);
    float A = amplitude * y * frequency / theta;
    return A * cos(theta * frequency + time * phase);
}

```

图 5

函数 **waveNormal** 根据函数 **dWavedx** 和 **dWavedy** 计算得到法线:

```

Vector3 waveNormal(float x, float y, float time){
    float dx = dWavedx(x, y, time);
    float dy = dWavedy(x, y, time);

    Vector3 n;
    n.x = -dx;
    n.y = 1.0;
    n.z = -dy;

    float l = sqrt(n.x*n.x + n.y*n.y + n.z*n.z);
    if (l != 0){
        n.x /= l;
        n.y /= l;
        n.z /= l;
    }
    else{
        n.x = 0;
        n.y = 1;
        n.z = 0;
    }

    return n;
}

```

图 6

注意：函数 waveNormal 返回值为向量，我们用一个结构体 Vector3 来表示一个向量：

```

struct Vector3{
    float x;
    float y;
    float z;
};

```

图 7

## 2. 水面建模

### 1) 构造网格

网格大小由常量 RESOLUTION 指定，水面网格大小定为  $\text{RESOLUTION} * (\text{RESOLUTION} + 1)$ 。

存储水面高度数据空间大小为：

$$3 * \text{RESOLUTION} * (\text{RESOLUTION} + 1)$$

用两个一维数组 surface, normal 分别存储水面高度点坐标和对应的法线向量：

```

static float surface[6 * RESOLUTION * (RESOLUTION + 1)];
static float normal[6 * RESOLUTION * (RESOLUTION + 1)];

```

图 8

### 2) 水面绘制

在计算得到所有网格点对应高度值后，链接相邻点三个点(x,y,H(x,y,t))绘制三角形，最终得到水面。

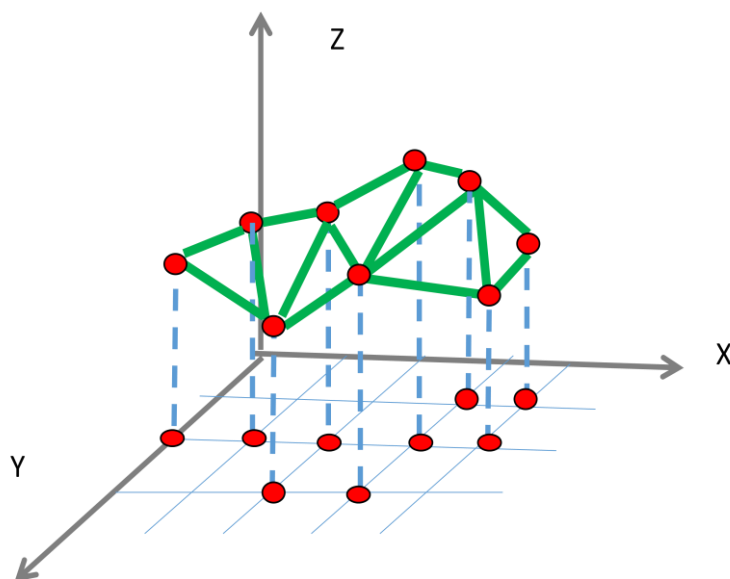


图 9

可以采用绘制基本图元的方法逐个绘制三角形，但是如果模型的顶点数或者三角面数过大（比如超过一万时），则程序运行速度会非常慢，根本就无法进行正常的调试。通常采用 **glDrawArray**、**glDrawElements** 这两个函数。这两个函数都能通过少数几条语句的调用实现大量数据的绘制，从而节省了函数调用的资源占用。这里采用 **glDrawArray** 函数。其函数原型：

```
void glDrawArrays(int mode, int first, int count)
```

函数参数：

**mode** ---- 绘制类型，包括如下类型：

- GL\_POINTS ---- 绘制独立的点
- GL\_LINE\_STRIP ---- 绘制一条线段
- GL\_LINE\_LOOP ---- 绘制一条封闭线段（首位相连）
- GL\_LINES ---- 绘制多条线段
- GL\_TRIANGLES ---- 绘制多个三角形（两两不相邻）
- GL\_TRIANGLE\_STRIP ---- 绘制多个三角形（两两相邻）
- GL\_TRIANGLE\_FAN ---- 以一个点为顶点绘制多个相邻的三角形

**first** ---- 多边形点的索引（标号）

**count** ---- 点的数目

描述：

该函数会从数组中找到第 **first** 个点，向后找到 **count** 个，用这些点来点依次相连绘制成多边形。

这里采用 GL\_TRIANGLE\_STRIP 模式：

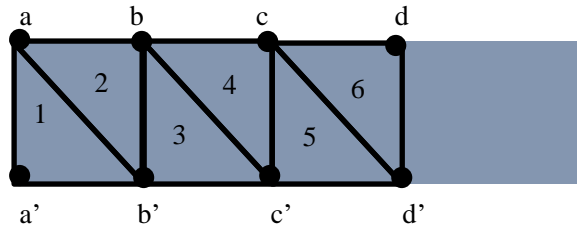


图 10

如图 10 所示的三角形两两相邻，存储时可以这样：

$a', a, b', b, c', c, d', d$

三角形 1 由顶点  $a', a, b'$  构成，三角形 2 由顶点  $a, b', b$  构成，依此类推。

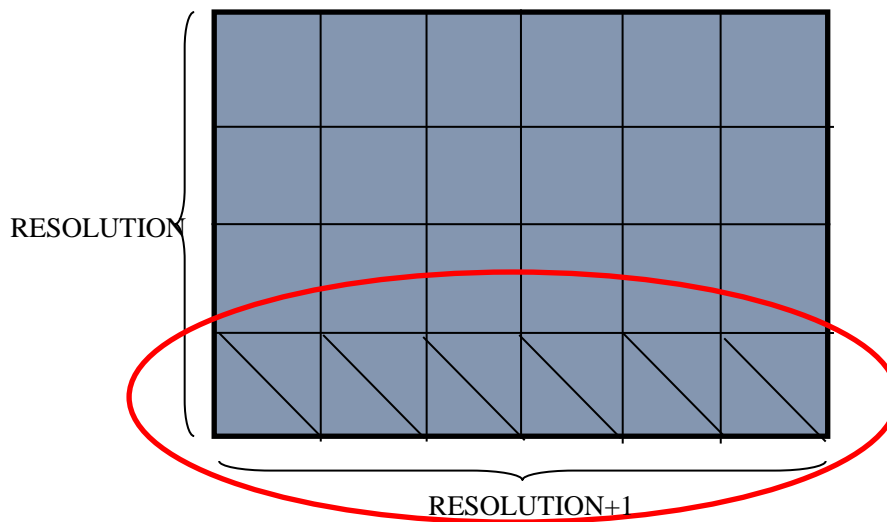


图 11

要绘制红圈中的三角形，需要存储顶点个数为  $2 * (\text{RESOLUTION} + 1)$ ，每个顶点坐标有三个分量，因此存储空间为：

$$6 * (\text{RESOLUTION} + 1)$$

绘制完整个水面，因此需要存储空间：

$$6 * \text{RESOLUTION} * (\text{RESOLUTION} + 1)$$

### 3. 其它程序模块

#### 1) 主函数



```

int main (int argc, char ** argv)
{
    //init GLUT and create window
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Water");

    // OpenGL settings
    InitGL();

    // register callbacks
    glutDisplayFunc(renderScene);
    glutReshapeFunc(changeSize);
    glutKeyboardFunc(Keyboard);

    // enter GLUT event processing cycle
    glutMainLoop ();

    return 0;
}

```

图 12

## 2) 初始参数设定

```

void InitGL() {
    glClearColor (0.0f, 0.0f, 0.0f, 0.0f);
    glClearDepth(1.0f);
    glEnable (GL_DEPTH_TEST);
    glDepthFunc (GL_LEQUAL);
    glHint (GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

    glLightfv(GL_LIGHT1, GL_AMBIENT, LightAmbient);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, LightDiffuse);
    glLightfv(GL_LIGHT1, GL_POSITION, LightPosition);
    glEnable(GL_LIGHT1);
    glEnable(GL_LIGHTING);

    glColorMaterial (GL_FRONT, GL_DIFFUSE);
    glEnable(GL_COLOR_MATERIAL);
}

```

图 13

## 3) 窗口响应函数

```

//Function called when the window is created or resized
void changeSize(int w, int h)
{
    if (h == 0) h = 1;

    float ratio = 1.0 * w / h;

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity ();

    glViewport (0, 0, w, h);

    gluPerspective (20, ratio, 0.1, 15);

    glMatrixMode(GL_MODELVIEW);
    glutPostRedisplay();
}

```

图 14

#### 4) 键盘控制函数

按键 l - 控制水面网格显示;

按键 n - 控制水面高度点处法线显示;

按键 q 或者 esc 键 - 退出程序。

```

// Function called when a key is hit
void Keyboard(unsigned char key, int x, int y)
{
    switch (key)
    {
        case 'q': case 27:
            exit (0);
            break;
        case 'l':
            wire_frame = !wire_frame;
            break;
        case 'n':
            normals = !normals;
            break;
    }
}

```

图 15

#### 5) 添加光照

```

//lighting
GLfloat LightAmbient[]=    { 0.9f, 0.9f, 0.9f, 1.0f };
GLfloat LightDiffuse[]=    { 0.9f, 0.9f, 0.9f, 1.0f };
GLfloat LightPosition[]=    { 1.0f, 1.0f, -0.5f, 0.0f };

```

图 16

至此，一个简单的水面模拟完成。

### (三) 添加鼠标控制

为了方便观察,我们将加入鼠标响应函数。通过捕捉鼠标移动,来控制观察物体的运动。

#### 1. 监测鼠标点击事件

##### 1) 注册回调函数

和键盘事件一样, GLUT 提供了响应鼠标点击事件的**注册回调函数**。函数名是 **glutMouseFunc**, 在主函数中被调用。原型如下:

```
void glutMouseFunc(void (*func)(int button, int state, int x, int y));
```

func - 鼠标点击事件的触发函数名。

在主函数中添加语句:

```
glutMouseFunc(Mouse);
```

##### 2) 响应函数

```
// Function called when a mouse button is hit
void Mouse(int button, int state, int x, int y)
{
    if (GLUT_LEFT_BUTTON == button)
        left_click = state;
    if (GLUT_RIGHT_BUTTON == button)
        right_click = state;
    xold = x;
    yold = y;
}
```

图 17

函数 **Mouse** 有 4 个参数:

##### 1) 第一个参数表示按下或释放了哪个键, 该参数有 3 个常量值选项:

GLUT\_LEFT\_BUTTON

GLUT\_MIDDLE\_BUTTON

GLUT\_RIGHT\_BUTTON

##### 2) 第二个参数表示事件触发时按键的状态, 例如是按下还是释放。可选值是:

GLUT\_DOWN

GLUT\_UP

当一个事件回调被带着 GLUT\_DOWN 状态触发的时候, 应用程序会自动断定 GLUT\_UP 的状态会在鼠标移离窗体的时候自动触发。

3) 剩下的两个参数是提供了鼠标相对于窗体客户区域左上角的  $x,y$  坐标.

## 2. 监测鼠标移动

### 1) 注册回调函数

GLUT 为应用程序提供鼠标移动监测的能力。有两类移动 GLUT 可以监测:**活跃移动**和**静默移动**。

- **活跃移动**是鼠标移动且鼠标键按下时触发。
- **静默移动**是鼠标移动且鼠标键没按下时触发。如果应用程序正在跟踪活动,在鼠标移动的期间每帧都会生成事件。

和之前一样, 需要注册 GLUT 回调函数来响应控制移动事件。GLUT 允许指定两个不同的函数: 一个跟踪静默移动, 一个跟踪活跃移动。原型如下:

```
void glutMotionFunc(void (*func) (int x,int y));
```

```
void glutPassiveMotionFunc(void (*func) (int x, int y));
```

func - 响应各自类型的移动的处理函数。

处理函数的参数是相对于窗体客户区域左上角的  $x,y$  坐标.

### 2) 响应函数

本程序中只用到活跃移动函数, 因此在主函数中添加:

```
glutMotionFunc(mouseMotion);
```

接下来实现 **mouseMotion** 函数。该函数功能如下:

- 按下鼠标左键时, 控制物体围绕  $x, y$  轴旋转;
- 按下鼠标右键, 控制物体沿着  $z$  轴平移, 起到放大缩小的作用;

具体实现如图 18 所示。

注意: 全局变量  $xold, yold$  分别记录鼠标在前一时刻位置。

```

// Function called when the mouse is moved
void mouseMotion(int x, int y)
{
    if (GLUT_DOWN == left_click)
    {
        rotate_y = rotate_y + (y - yold) / 5.0;
        rotate_x = rotate_x + (x - xold) / 5.0;
        if (rotate_y > 90)
            rotate_y = 90;
        if (rotate_y < -90)
            rotate_y = -90;
        glutPostRedisplay ();
    }
    if (GLUT_DOWN == right_click)
    {
        translate_z = translate_z + (yold - y) / 50.;
        if (translate_z < 0.5)
            translate_z = 0.5;
        if (translate_z > 10)
            translate_z = 10;
        glutPostRedisplay ();
    }
    xold = x;
    yold = y;
}

```

图 18

#### (四) 纹理添加

为了让水面更生动，我们为水面添加如下纹理：

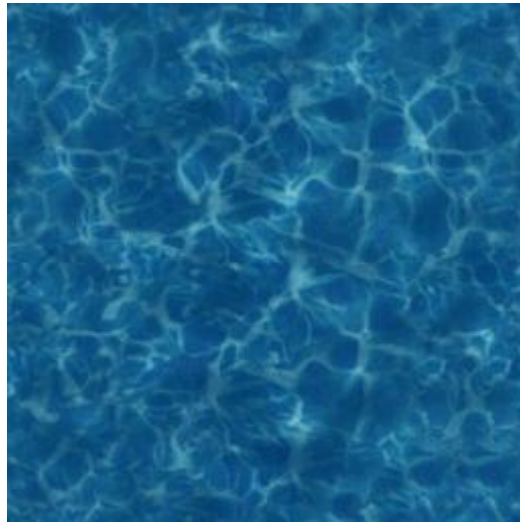


图 19

##### 1. 载入纹理

载入纹理图片，同时设置纹理参数。这里我们并不需要为手工指定纹理坐标，而是由OpenGL自动为每个顶点分配坐标。这个任务由函数`glTexGen*()`来完成。

```
int LoadGLTextures() {
    caustic_texture = SOIL_load_OGL_texture
    (
        "reflection.jpg",
        SOIL_LOAD_AUTO,
        SOIL_CREATE_NEW_ID,
        SOIL_FLAG_INVERT_Y
    );

    if (caustic_texture == 0)
        return false;

    glBindTexture(GL_TEXTURE_2D, caustic_texture);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
    glEnable(GL_TEXTURE_GEN_S);
    glEnable(GL_TEXTURE_GEN_T);
    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
    glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);

    return true;
}
```

图 20

##### 2. 修改初始化函数

修改 `InitGL` 函数：

```

int InitGL() {
    if (!LoadGLTextures())
        return false;

    glClearColor (0.0f, 0.0f, 0.0f, 0.0f);
    glClearDepth(1.0f);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

    return true;
}

```

图 21

另外，不要忘记在 renderScene 函数中开启纹理映射：

```
glEnable(GL_TEXTURE_2D);
```

最后完成效果如图 22 所示：

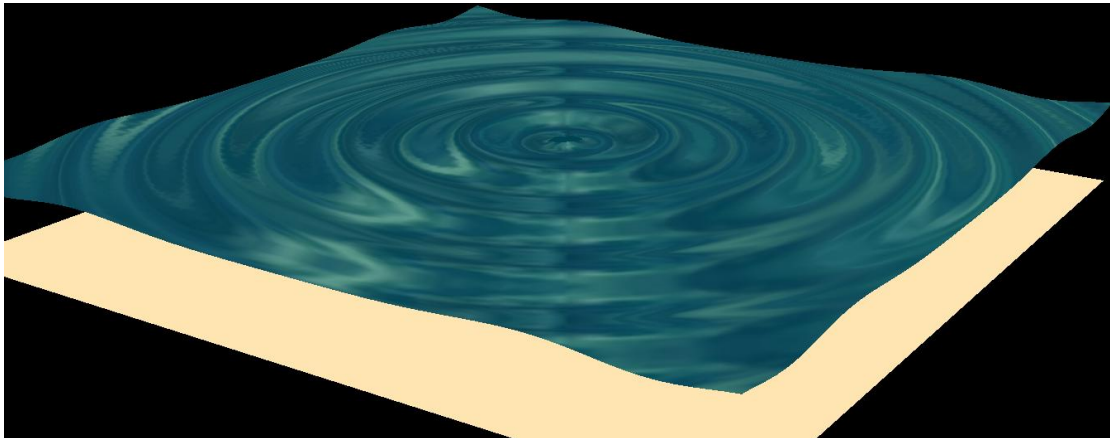


图 22

## (五) 水面模拟 - 多个振源组合

尝试添加多个振源，创建复杂的水面，如图 23 所示：

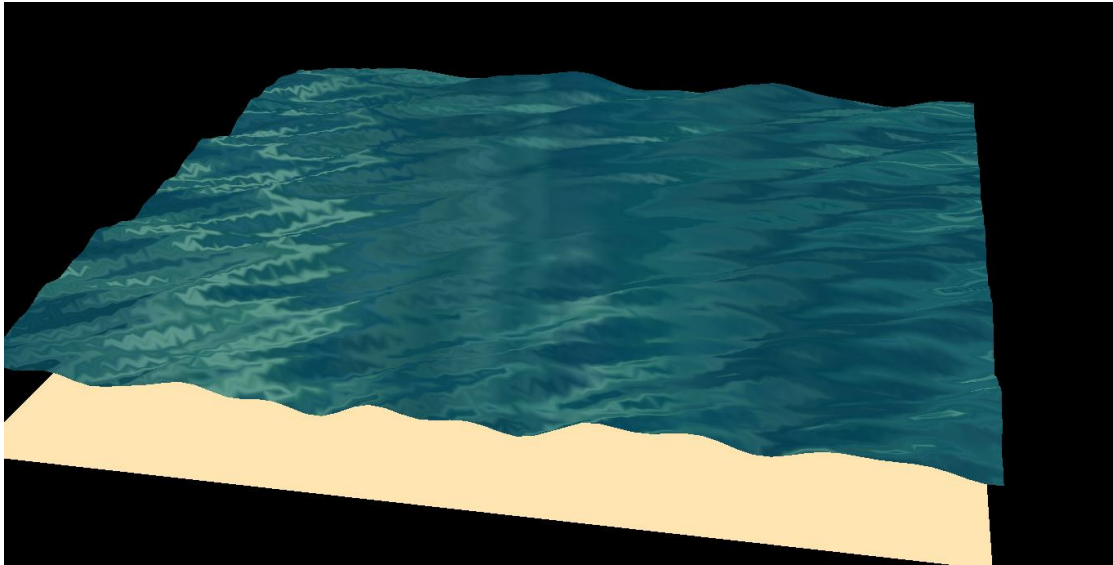


图 23

### 1. 水面高度函数

适当修改水面高度函数 **waveHeight**:

```
float dot(int i, float x, float y){
    float xc = x - centers[i][0];
    float yc = y - centers[i][1];
    return sqrt(xc * xc + yc * yc);
}

float wave(int i, float x, float y, float time){
    float frequency = 2 * PI/wavelength[i];
    float phase = speed[i] * frequency;
    float theta = dot(i,x,y);

    return amplitude[i] * sin(theta * frequency + time * phase);
}

float waveHeight(float x, float y, float time){
    float height = 0.0;
    for (int i = 0; i < numWaves; i++){
        height += wave(i, x, y, time);
    }
    return height;
}
```

图 24

水面高度函数 **wave** 计算第 *i* 个振源，在位置(x,y)处的水面高度，**waveHeight** 将每个振源算得的结果累加就得到最终水面高度，其中全局变量 **numWaves** 指定了振源个数。

### 2. 法线计算

函数 **dWavedx** 和函数 **dWavedy** 分别计算了第 *i* 个振源在 *x* 和 *y* 方向上偏导数：



```

float dWavedx(int i, float x, float y, float time){
    float frequency = 2 * PI / wavelength[i];
    float phase = speed[i] * frequency;
    float theta = dot(i,x, y);
    float A = amplitude[i] * x * frequency / theta;
    return A * cos(theta * frequency + time * phase);
}

float dWavedy(int i, float x, float y, float time){
    float frequency = 2 * PI / wavelength[i];
    float phase = speed[i] * frequency;
    float theta = dot(i, x, y);
    float A = amplitude[i] * y * frequency / theta;
    return A * cos(theta * frequency + time * phase);
}

```

图 25

函数 **waveNormal** 也需要相应修改:

```

Vector3 waveNormal(float x, float y, float time){
    float dx = 0.0;
    float dy = 0.0;
    for(int i = 0; i < numWaves; i++){
        dx += dWavedx(i, x, y, time);
        dy += dWavedy(i, x, y, time);
    }
    Vector3 n;
    n.x = -dx;
    n.y = 1.0;
    n.z = -dy;

    float l = sqrt(n.x*n.x + n.y*n.y + n.z*n.z);
    if (l != 0){
        n.x = n.x/l;
        n.y = n.y/l;
        n.z = n.z/l;
    }
    else
    {
        n.x = 0;
        n.y = 1;
        n.z = 0;
    }

    return n;
}

```

图 26

### (六) 变化波浪模型

尝试变化波浪波形，采用如下模型：

$$H_i(x, y, t) = 2A_i \times \left( \frac{\sin(D_i \cdot (x, y) \times w_i + t \times \varphi_i) + 1}{2} \right)^k$$

新模型X方向和Y方向偏导数分别为：

$$\frac{\partial}{\partial x}(H_i(x, y, t)) = k \times D_i \cdot x \times w_i \times A_i \times \left( \frac{\sin(D_i \cdot (x, y) \times w_i + t \times \varphi_i) + 1}{2} \right)^{k-1}$$

$$\times \cos(D_i \cdot (x, y) \times w_i + t \times \varphi_i)$$

$$\frac{\partial}{\partial y}(H_i(x, y, t)) = k \times D_i \cdot y \times w_i \times A_i \times \left( \frac{\sin(D_i \cdot (x, y) \times w_i + t \times \varphi_i) + 1}{2} \right)^{k-1}$$

$$\times \cos(D_i \cdot (x, y) \times w_i + t \times \varphi_i)$$

## (七) Gerstner Waves

尝试 Gerstner Wave 模型：

Gerstner wave函数为：

$$\mathbf{P}(x, y, t) = \begin{pmatrix} x + \sum (Q_i A_i \times \mathbf{D}_i \cdot \mathbf{x} \times \cos(w_i \mathbf{D}_i \cdot (x, y) + \varphi_i t)) \\ y + \sum (Q_i A_i \times \mathbf{D}_i \cdot \mathbf{y} \times \cos(w_i \mathbf{D}_i \cdot (x, y) + \varphi_i t)) \\ \sum (A_i \times \sin(w_i \mathbf{D}_i \cdot (x, y) + \varphi_i t)) \end{pmatrix}$$

–  $Q_i$ 为控制波浪陡峭程度的参数；

法线的计算：

$$\mathbf{B} = \begin{pmatrix} 1 - \sum (Q_i \times D_i \cdot x^2 \times WA \times S(\cdot)), \\ -\sum (Q_i \times D_i \cdot x \times D_i \cdot y \times WA \times S(\cdot)), \\ \sum (D_i \cdot x \times WA \times C(\cdot)) \end{pmatrix}, \quad \mathbf{T} = \begin{pmatrix} -\sum (Q_i \times D_i \cdot x \times D_i \cdot y \times WA \times S(\cdot)), \\ 1 - \sum (Q_i \times D_i \cdot y^2 \times WA \times S(\cdot)), \\ \sum (D_i \cdot y \times WA \times C(\cdot)) \end{pmatrix}$$

$$\mathbf{N} = \begin{pmatrix} -\sum (D_i \cdot x \times WA \times C(\cdot)), \\ -\sum (D_i \cdot y \times WA \times C(\cdot)), \\ 1 - \sum (Q_i \times WA \times S(\cdot)) \end{pmatrix}$$

其中：

- $WA = w_i \times A_i$
- $S(\cdot) = \sin(w_i \times \mathbf{D}_i \cdot \mathbf{P} + \varphi_i t)$
- $C(\cdot) = \cos(w_i \times \mathbf{D}_i \cdot \mathbf{P} + \varphi_i t)$

## 三、实验内容

1. 实现一个基础的水面模拟 – 单振源；
2. 为 1) 中水面模拟程序添加光照和键盘响应函数；
3. 为 2) 中水面模拟程序添加鼠标响应控制函数；
4. 为 3) 中水面模拟程序添加纹理映射；
5. 为 4) 中水面模拟程序添加多个振源；
6. 修改 5) 中水面模拟程序，将波浪模型改为指数控制的模型：

$$H_i(x, y, t) = 2A_i \times \left( \frac{\sin(D_i \cdot (x, y) \times w_i + t \times \varphi_i) + 1}{2} \right)^k$$

7. （选作）修改 5) 中水面模拟程序，将波浪模型改为 Gerstner Wave 模型：

Gerstner wave函数为:

$$P(x, y, t) = \begin{pmatrix} x + \sum (Q_i A_i \times D_i \cdot x \times \cos(w_i D_i \cdot (x, y) + \varphi_i t)), \\ y + \sum (Q_i A_i \times D_i \cdot y \times \cos(w_i D_i \cdot (x, y) + \varphi_i t)), \\ \sum (A_i \times \sin(w_i D_i \cdot (x, y) + \varphi_i t)) \end{pmatrix}$$

–  $Q_i$ 为控制波浪陡峭程度的参数;

## 四、实验步骤

1. 明确项目需求;
2. 编写代码;
3. 编译代码;
4. 测试程序;
5. 根据测试结果对程序进行调试改进。