```c
/* Simple Demo for GLSL www.lighthouse3d.com - tweaked by andy  */

#include "GL/glew.h"

#ifdef __linux__
#include <GL/glut.h>
#endif

#ifdef __APPLE__
#include <glut.h>
#endif

#ifdef _WIN32
#include "glut.h"
#endif

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

char normalData[800000];
GLuint normaltex;
GLuint texNumNormal;

float mouseWindowX = 0.0;
float mouseWindowY = 0.0;

int windowWidth = 320;
int windowHeight = 320;

float bSize    = 0.15;
float bDensity = 16.0;

GLuint p;

float lpos[4] = {1.0, 0.5, 1.0, 0.0};

//////////////////////////////////////////////////////////////////

char *textFileRead(char *fn)
{
   FILE *fp;
   char *content = NULL;

   int count=0;

   if (fn != NULL) {
        fp = fopen(fn,"rt");

        if (fp != NULL) {

            fseek(fp, 0, SEEK_END);
            count = ftell(fp);
            rewind(fp);

         if (count > 0) {
            content = (char *)malloc(sizeof(char) * (count+1));
            count = fread(content,sizeof(char),count,fp);
            content[count] = '\0';
         }
         fclose(fp);
      }
   }

   if (content == NULL)
        {
        fprintf(stderr, "ERROR: could not load in file %s\n", fn);
        exit(1);
        }
    return content;
}

//////////////////////////////////////////////////////////////////

void changeSize(int w, int h) {

    // Prevent a divide by zero, when window is too short
    // (you cant make a window of zero width).
    if(h == 0)
        h = 1;

    float ratio = 1.0* w / h;

    // Reset the coordinate system before modifying
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    // Set the viewport to be the entire window
    glViewport(0, 0, w, h);

    windowWidth = w;
    windowHeight = h;

   // Set the correct perspective.
   gluPerspective(45, ratio, 1, 1000);
   glMatrixMode(GL_MODELVIEW);
}

//////////////////////////////////////////////////////////////////

void printShaderLog(GLuint obj) {
   GLint infoLogLength = 0;
   GLsizei charsWritten  = 0;
   GLchar *infoLog;

   glGetShaderiv(obj, GL_INFO_LOG_LENGTH, &infoLogLength);

   if (infoLogLength > 0){
      infoLog = (char *) malloc(infoLogLength);
      glGetShaderInfoLog(obj, infoLogLength, &charsWritten, infoLog);
      printf("%s\n",infoLog);
      free(infoLog);
      }
   }

void printProgramLog(GLuint obj) {
   GLint infoLogLength = 0;
   GLsizei charsWritten  = 0;
   GLchar *infoLog;

   glGetProgramiv(obj, GL_INFO_LOG_LENGTH, &infoLogLength);

   if (infoLogLength > 0){
      infoLog = (char *) malloc(infoLogLength);
      glGetProgramInfoLog(obj, infoLogLength, &charsWritten, infoLog);
      printf("%s\n",infoLog);
      free(infoLog);
      }
   }

//////////////////////////////////////////////////////////////////

void PassiveMouseMotion(int x, int y)
{
    if (x<0)
        mouseWindowX = -1;
    else if (x>windowWidth)
        mouseWindowX = 1;
    else
        mouseWindowX = 2* (x / (float) windowWidth) - 1.0;

    if (y<0)
        mouseWindowY = -1;
    else if (y>windowHeight)
        mouseWindowY = 1;
    else
        mouseWindowY = 2* (y / (float) windowHeight) - 1.0;
    }

//////////////////////////////////////////////////////////////////

void processNormalKeys(unsigned char key, int x, int y) {
    if (key == 27)
      exit(0);

    // size
    if (key == '1')
        bSize += 0.02;
    if (key == '2')
        bSize -= 0.02;

    // density
    if (key == '3')
        bDensity += 1.0;
    if (key == '4')
        bDensity -= 1.0;
}

//////////////////////////////////////////////////////////////////

GLuint setShaders(char * vert, char * frag) {
    GLuint v,f, pro;
   char *vs,*fs;

   v = glCreateShader(GL_VERTEX_SHADER);
   f = glCreateShader(GL_FRAGMENT_SHADER);

   vs = textFileRead(vert);
   fs = textFileRead(frag);

   const char * vv = vs;
   const char * ff = fs;

   glShaderSource(v, 1, &vv,NULL);
   glShaderSource(f, 1, &ff,NULL);

   free(vs);free(fs);

   glCompileShader(v);
   glCompileShader(f);

   printShaderLog(v);
   printShaderLog(f);

   pro = glCreateProgram();
   glAttachShader(pro,v);
   glAttachShader(pro,f);

   glLinkProgram(pro);
   printProgramLog(pro);
```

```
202:
203:    return(pro);
204: }
205:
206: ////////////////////////////////////////////////////////////////////
207:
208: void renderScene(void) {
209:
210:    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
211:
212:    glLoadIdentity();
213:    gluLookAt(  0.0f, 3.0f, 6.0f,
214:                0.0f, 0.0f,-1.0f,
215:                0.0f, 1.0f, 0.0f);
216:
217:     glLightfv(GL_LIGHT0, GL_POSITION, lpos);
218:
219:     glRotatef(mouseWindowX*140+180,0,1,0);
220:     glRotatef(mouseWindowY*90+22,1,0,0);
221:
222:     glUseProgram(p);
223:
224:     GLint texLoc, lightLoc, bumpSizeLoc, bumpDensityLoc;
225:     texLoc = glGetAttribLocation(p, "Tangent");
226:     glVertexAttrib3f(texLoc, 1.0, 0.0, 0.0);
227:
228:     lightLoc = glGetUniformLocation(p, "LightPosition");
229:     glUniform3f(lightLoc, 1.0, 1.0, 4.0);
230:
231:     bumpSizeLoc = glGetUniformLocation(p, "BumpSize");
232:     glUniform1f(bumpSizeLoc, bSize);
233:
234:     bumpDensityLoc = glGetUniformLocation(p, "BumpDensity");
235:     glUniform1f(bumpDensityLoc, bDensity);
236:
237:     glNormal3f(0.0f, 0.0f, -1.0f );
238:
239:     glBegin( GL_POLYGON );
240:         glTexCoord2f( 0.0f, 1.0f );
241:         glVertex3f( -2.0f, -2.0f, 0.0f );
242:
243:         glTexCoord2f( 1.0f, 1.0f );
244:         glVertex3f(  2.0f, -2.0f, 0.0f );
245:
246:         glTexCoord2f( 1.0f, 0.0f );
247:         glVertex3f(  2.0f,  2.0f, 0.0f );
248:
249:         glTexCoord2f( 0.0f, 0.0f );
250:         glVertex3f( -2.0f,  2.0f, 0.0f );
251:     glEnd();
252:
253:     glutSwapBuffers();
254: }
255:
256: ////////////////////////////////////////////////////////////////////
257:
258: int main(int argc, char **argv) {
259:    glutInit(&argc, argv);
260:    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
261:    glutInitWindowPosition(100,100);
262:    glutInitWindowSize(windowWidth, windowHeight);
263:    glutCreateWindow("GLSL");
264:
265:    glutDisplayFunc(renderScene);
266:    glutIdleFunc(renderScene);
267:    glutReshapeFunc(changeSize);
268:    glutKeyboardFunc(processNormalKeys);
269:    glutPassiveMotionFunc(PassiveMouseMotion);
270:
271:    glEnable(GL_DEPTH_TEST);
272:    glClearColor(0.0,0.0,0.0,1.0);
273:
274:
275:    glewInit();
276:    if (GLEW_ARB_vertex_shader && GLEW_ARB_fragment_shader)
277:         printf("Ready for GLSL\n");
278:    else {
279:         printf("No GLSL support\n");
280:         exit(1);
281:     }
282:
283:     p = setShaders("./bump2.vert", "./bump2.frag");
284:
285:    glutMainLoop();
286:    return 0;
287: }
288:
289: ////////////////////////////////////////////////////////////////////
290: // Fragment shader for procedural bumps
291: // Authors: Randi Rost, John Kessenich
292: // Copyright (c) 2002-2005 3Dlabs Inc. Ltd.
293: // See 3Dlabs-License.txt for license information
294:
295: varying vec3 LightDir;
296: varying vec3 EyeDir;
297:
298: vec3 SurfaceColor = vec3(0.7, 0.6, 0.18);
299:
300: uniform float BumpDensity;
301: //float BumpDensity = 16.0;
302:
303: float SpecularFactor = 0.5;
304:
305: uniform float BumpSize;
306: //float BumpSize    = 0.15;
307:
308: void main (void)
309: {
310:    vec3 litColor;
311:    vec2 c = BumpDensity * gl_TexCoord[0].st;
312:    vec2 p = fract(c) - vec2 (0.5);
313:
314:    float d, f;
315:    d = p.x * p.x + p.y * p.y;
316:    f = 1.0 / sqrt(d + 1.0);
317:
318:    // if we are outside the area of a bump then
319:    // the normal points straight out Z w/ length 1
320:    if (d >= BumpSize)
321:       {
322:       p = vec2(0.0);
323:       f = 1.0;
324:       }
325:
326:    // create a new normal of length 1 for this fragment
327:    vec3 normDelta = vec3 (p.x, p.y, 1.0) * f;
328:
329:    litColor = SurfaceColor * max(dot(normDelta, LightDir), 0.0);
330:    vec3 reflectDir = reflect(LightDir, normDelta);
331:
332:    float spec = max(dot(EyeDir, reflectDir), 0.0);
333:    spec = pow(spec, 6.0);
334:    spec *= SpecularFactor;
335:    litColor = min(litColor + spec, vec3 (1.0));
336:
337:    gl_FragColor = vec4 (litColor, 1.0);
338: }////////////////////////////////////////////////////////////////////
339: // Vertex shader for procedural bumps
340: // Authors: Randi Rost, John Kessenich
341: // Copyright (c) 2002-2005 3Dlabs Inc. Ltd.
342: // See 3Dlabs-License.txt for license information
343:
344: varying vec3 LightDir;
345: varying vec3 EyeDir;
346:
347: uniform vec3 LightPosition;
348:
349: attribute vec3 Tangent;
350:
351: void main(void)
352: {
353:    EyeDir = vec3 (gl_ModelViewMatrix * gl_Vertex);
354:
355:    gl_Position = ftransform();
356:    gl_TexCoord[0] = gl_MultiTexCoord0;
357:
358:    // convert normal and tangent (from main program) into eye space
359:    vec3 n = normalize(gl_NormalMatrix * gl_Normal);
360:    vec3 t = normalize(gl_NormalMatrix * Tangent);
361:
362:    // compute b in eye space
363:    vec3 b = cross(n, t);
364:
365:    // convert light direction from eye space to tangent space
366:    vec3 v;
367:    v.x = dot(LightPosition, t);
368:    v.y = dot(LightPosition, b);
369:    v.z = dot(LightPosition, n);
370:    LightDir = normalize(v);
371:
372:    // convert eye direction from eye space to tangent space
373:    v.x = dot(EyeDir, t);
374:    v.y = dot(EyeDir, b);
375:    v.z = dot(EyeDir, n);
376:    EyeDir = normalize(v);
377: }
```