

```
#include<stdio.h>

#include<stdlib.h>

#include<GL/gl.h>

#include<GL/glu.h>

#include<SDL.h>

#define GL_ARRAY_BUFFER_ARB 0x8892

#define GL_STATIC_DRAW_ARB 0x88E4

typedef void (APIENTRY * PFNGLBINDBUFFERARBPROC) (GLenum target, GLuint buffer);

typedef void (APIENTRY * PFNGLDELETEBUFFERSARBPROC) (GLsizei n, const GLuint *buffers);

typedef void (APIENTRY * PFNGLGENBUFFERSARBPROC) (GLsizei n, GLuint *buffers);

typedef void (APIENTRY * PFNGLBUFFERDATAARBPROC) (GLenum target, int size, const GLvoid *data,
GLenum usage);

PFNGLGENBUFFERSARBPROC glGenBuffersARB = NULL;

PFNGLBINDBUFFERARBPROC glBindBufferARB = NULL;

PFNGLBUFFERDATAARBPROC glBufferDataARB = NULL;

PFNGLDELETEBUFFERSARBPROC glDeleteBuffersARB = NULL;

typedef struct { float x,y,z; } quad_vert;

typedef struct { float x,y,z; } quad_norm;

typedef struct { float x,y,z; } tri_vert;

typedef struct { float x,y,z; } tri_norm;

struct { int tri_verts, tri_norms, quad_verts, quad_norms, models;
```

```
unsigned int tri_vbo, quad_vbo, quad_vbon, tri_vbon;} number;
```

```
struct window{
```

```
    int width;
```

```
    int height;
```

```
    int bpps;
```

```
    int flags;
```

```
};
```

```
int mousex;
```

```
int mousey;
```

```
char *key;
```

```
SDL_Event event;
```

```
GLfloat i;
```

```
GLfloat n;
```

```
void input(struct window *win){
```

```
    SDL_Event event;
```

```
    while ( SDL_PollEvent(&event) ) {
```

```
        switch (event.type) {
```

```
            case SDL_MOUSEMOTION:
```

```
                mousex=event.motion.x;
```

```
        mousey=event.motion.y;

                break;

case SDL_MOUSEBUTTONDOWN:

glDeleteBuffersARB(1, &number.tri_vbon);

glDeleteBuffersARB(1, &number.tri_vbo);

glDeleteBuffersARB(1, &number.quad_vbo);

glDeleteBuffersARB(1, &number.quad_vbon);

        SDL_Quit();

        exit(0);

break;

case SDL_KEYDOWN:

key=SDL_GetKeyName(event.key.keysym.sym);

switch( event.key.keysym.sym ){

        case SDLK_LEFT:

                i--;

                break;

        case SDLK_RIGHT:

                i++;

                break;

        case SDLK_UP:

                n++;
```

```
        break;
    case SDLK_DOWN:
        n--;
        break;
    default:
        break;}

}

}
```

```
/* Arrays for Quads/Triangles */
```

```
quad_vert **quad_v;
```

```
quad_norm **quad_n;
```

```
tri_vert **tri_v;
```

```
tri_norm **tri_n;
```

```
int LoadModel(){
```

```
#define FILE_NAME "object1.c"
```

```
FILE *fp=fopen(FILE_NAME, "r");
```

```
char *txt = malloc(1024);

number.models=2;

int counter;

tri_n = malloc(number.models * sizeof *tri_n);
tri_v = malloc(number.models * sizeof *tri_v);
quad_v = malloc(number.models * sizeof *quad_v);
quad_n = malloc(number.models * sizeof *quad_n);

/* TRIANGLES */
while(!feof(fp)){
fscanf(fp, "%s",txt);

if(strcmp(txt,"TRI")==0){

//printf("\nTriangles %d:\n",t);
for(counter=0; counter!=3;counter++)
{

/* Normal 1 */
fscanf(fp, "%s",txt);
if(strcmp(txt,"n")==0){

number.tri_norms++;
```

```

tri_n[number.models-1] = realloc(tri_n[number.models-1], (number.tri_norms * sizeof
*tri_n[number.models-1]));

fscanf(fp, "%f %f %f",&tri_n[number.models-1][number.tri_norms-1].x,&tri_n[number.models-
1][number.tri_norms-1].y,&tri_n[number.models-1][number.tri_norms-1].z);

//printf("Normal %d: %.10f %.10f %.10f\n",number.tri_norms-1,tri_n[number.models-
1][number.tri_norms-1].x,tri_n[number.models-1][number.tri_norms-1].y,tri_n[number.models-
1][number.tri_norms-1].z);

}

/* Vertex 1 */
fscanf(fp, "%s",txt);
if(strcmp(txt,"v")==0){

number.tri_verts++;

tri_v[number.models-1] = realloc(tri_v[number.models-1], (number.tri_verts * sizeof
*tri_v[number.models-1]));

fscanf(fp, "%f %f %f",&tri_v[number.models-1][number.tri_verts-1].x,&tri_v[number.models-
1][number.tri_verts-1].y ,&tri_v[number.models-1][number.tri_verts-1].z);

// printf("Vertex %d: %.10f %.10f %.10f\n",number.tri_verts-1,tri_v[number.models-
1][number.tri_verts-1].x,tri_v[number.models-1][number.tri_verts-1].y,tri_v[number.models-
1][number.tri_verts-1].z);

}

}}

```

```

}

fclose(fp);
fopen(FILE_NAME, "r");

/* QUAD */
while(!feof(fp)){
fscanf(fp, "%s",txt);

if(strcmp(txt,"QUAD")==0){

//printf("\nQuads:\n");
for(counter=0; counter!=4;counter++)
{

/* Normal 1 */
fscanf(fp, "%s",txt);
if(strcmp(txt,"n")==0){

number.quad_norms++;

quad_n[number.models-1] = realloc(quad_n[number.models-1], (number.quad_norms * sizeof
*quad_n[number.models-1]));

fscanf(fp, "%f %f %f\n", &quad_n[number.models-1][number.quad_norms-1].x,
&quad_n[number.models-1][number.quad_norms-1].y, &quad_n[number.models-
1][number.quad_norms-1].z);

```

```

//printf("Normal %d: %.10f %.10f %.10f\n",number.quad_norms-1, quad_n[number.models-1][number.quad_norms-1].x, quad_n[number.models-1][number.quad_norms-1].y, quad_n[number.models-1][number.quad_norms-1].z);

}

/* Vertex 1 */

fscanf(fp, "%s",txt);

if(strcmp(txt,"v")==0){

number.quad_verts++;

quad_v[number.models-1] = realloc(quad_v[number.models-1], (number.quad_verts * sizeof *quad_v[number.models-1]));

fscanf(fp, "%f %f %f\n", &quad_v[number.models-1][number.quad_verts-1].x, &quad_v[number.models-1][number.quad_verts-1].y, &quad_v[number.models-1][number.quad_verts-1].z);

//printf("Normal %d: %.10f %.10f %.10f\n",number.quad_verts-1, quad_v[number.models-1][number.quad_verts-1].x, quad_v[number.models-1][number.quad_verts-1].y, quad_v[number.models-1][number.quad_verts-1].z);

}

}}

}

free(txt);

```

```

fclose(fp);

}

int VBO(){

glGenBuffersARB = (PFNGLGENBUFFERSARBPROC) SDL_GL_GetProcAddress("glGenBuffersARB");
glBindBufferARB = (PFNGLBINDBUFFERARBPROC) SDL_GL_GetProcAddress("glBindBufferARB");
glBufferDataARB = (PFNGLBUFFERDATAARBPROC) SDL_GL_GetProcAddress("glBufferDataARB");
glDeleteBuffersARB = (PFNGLDELETEBUFFERSARBPROC)
SDL_GL_GetProcAddress("glDeleteBuffersARB");

/* Triangles Normals */

glGenBuffersARB(1,&number.tri_vbon);

glBindBufferARB(GL_ARRAY_BUFFER_ARB, number.tri_vbon);

glBufferDataARB(GL_ARRAY_BUFFER_ARB, number.tri_norms*3*sizeof(tri_vert), (void *)tri_n[1],
GL_STATIC_DRAW_ARB );

/* Triangles Vertices */

glGenBuffersARB(1,&number.tri_vbo);

glBindBufferARB(GL_ARRAY_BUFFER_ARB, number.tri_vbo);

glBufferDataARB(GL_ARRAY_BUFFER_ARB, number.tri_verts*3*sizeof(tri_vert), (void *)tri_v[1],
GL_STATIC_DRAW_ARB );

/* Quads Normals */

glGenBuffersARB(1,&number.quad_vbon);

glBindBufferARB(GL_ARRAY_BUFFER_ARB, number.quad_vbon);

```

```
glBufferDataARB(GL_ARRAY_BUFFER_ARB, number.quad_norms*4*sizeof(quad_norm),(void
*)quad_n[1] , GL_STATIC_DRAW_ARB);

/* Quads Vertices */

glGenBuffersARB(1,&number.quad_vbo);

glBindBufferARB(GL_ARRAY_BUFFER_ARB, number.quad_vbo);

glBufferDataARB(GL_ARRAY_BUFFER_ARB, number.quad_verts*4*sizeof(quad_vert),(void *)quad_v[1] ,
GL_STATIC_DRAW_ARB );

}

void GenWindow(struct window *win){

win->width=1024;

win->height=768;

win->bpps=24;

SDL_Init(SDL_INIT_VIDEO);

SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER,1);

SDL_GL_SetAttribute(SDL_GL_DEPTH_SIZE,24);

SDL_SetVideoMode(win->width,win->height,win->bpps,SDL_OPENGL|SDL_FULLSCREEN);

glClearColor( 0, 0, 0, 0);
```

```
glViewport(0,0,win->width,win->height);
```

```
glMatrixMode( GL_PROJECTION );
```

```
glLoadIdentity();
```

```
printf("\n%d x %d @ %d Window Created.\n", win->width, win->height, win->bpps);
```

```
}
```

```
void scene(struct window *win){
```

```
    register float ratio = (float) win->width / (float) win->height;
```

```
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
```

```
glMatrixMode( GL_MODELVIEW );
```

```
glLoadIdentity();
```

```
gluPerspective(50.0, ratio, 1.0, 10000);
```

```
GLfloat lightAmbient[] = {1.0f, 0.0f, 0.0f, 1.0f} ;
```

```
glEnable(GL_LIGHTING) ;

glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lightAmbient) ;

GLfloat lightPosition[] = {9.7f, 9.0f, 0.0f, 0.0f} ;

GLfloat lightDiffuse[] = {1.0f, 1.0f, 1.0f, 1.0f} ;

GLfloat lightSpecular[] = {1.0f, 1.0f, 1.0f, 1.0f} ;

glEnable(GL_LIGHT0) ;

glLightfv(GL_LIGHT0, GL_POSITION,lightPosition) ;

glLightfv(GL_LIGHT0, GL_AMBIENT,lightAmbient) ;

glLightfv(GL_LIGHT0, GL_DIFFUSE,lightDiffuse) ;

glLightfv(GL_LIGHT0, GL_SPECULAR,lightSpecular) ;

glEnable(GL_BLEND);

glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

glEnable(GL_LINE_SMOOTH);

glShadeModel(GL_SMOOTH);

glDisable(GL_CULL_FACE);

glEnable(GL_DEPTH_TEST);

glHint(GL_PERSPECTIVE_CORRECTION_HINT,GL_NICEST);

glHint(GL_POINT_SMOOTH_HINT,GL_NICEST);

glTranslatef(i,n,-3.0f);
```

```
glRotatef(1.0f+mousex,0.0f,1.0f,0.0f);
```

```
glRotatef(1.0f+mousey,1.0f,0.0f,0.0f);
```

```
/* triangles */
```

```
glEnableClientState(GL_NORMAL_ARRAY);
```

```
glBindBufferARB(GL_ARRAY_BUFFER_ARB, number.tri_vbon);
```

```
glNormalPointer(GL_FLOAT, 0, (char *)NULL);
```

```
glEnableClientState(GL_VERTEX_ARRAY);
```

```
glBindBufferARB(GL_ARRAY_BUFFER_ARB, number.tri_vbo);
```

```
glVertexPointer(3, GL_FLOAT,0,(char *)NULL);
```

```
glDrawArrays(GL_TRIANGLES,0,number.tri_verts);
```

```
glDisableClientState(GL_VERTEX_ARRAY);
```

```
/* quads */
```

```
glBindBufferARB(GL_ARRAY_BUFFER_ARB, number.quad_vbon);
```

```
glNormalPointer(GL_FLOAT, 0, (char *)NULL);
```

```
glEnableClientState(GL_VERTEX_ARRAY);
```

```
glBindBufferARB(GL_ARRAY_BUFFER_ARB, number.quad_vbo);
```

```
glVertexPointer(3, GL_FLOAT,0,(char *)NULL);
```

```
glDrawArrays(GL_QUADS,0,number.quad_verts);
```

```
glDisableClientState(GL_NORMAL_ARRAY);
```

```
glDisableClientState(GL_VERTEX_ARRAY);
```

```
SDL_GL_SwapBuffers();
```

```
}
```

```
int main(int argc, char* argv[]){
```

```
    struct window win;
```

```
    LoadModel();
```

```
    GenWindow(&win);
```

```
    VBO();
```

```
    SDL_EnableKeyRepeat(100,SDL_DEFAULT_REPEAT_INTERVAL);
```

```
    SDL_ShowCursor(SDL_DISABLE);
```

```
    do{
```

```
        input(&win);
```

```
        scene(&win);
```

```
}while(1);
```

```
return 0;}
```