ДОДАТОК А

Графічний матеріл атестаційної роботи

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
Факультет комп'ютерної інженерії та управління
Кафедра ЕОМ

Моделювання рельєфу в системах візуалізації для авіаційних тренажерів

Виконав:
Студент групи КСМм-19-1
Остапенко Т.О.
Науковий керівник:
Мовсесян Я. С.

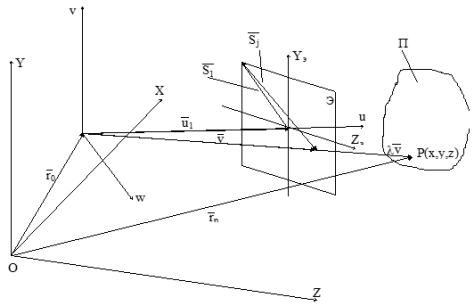
Мета та постановка завдання

Мета: підвищення реалістичності зображення в системах візуалізації, з використанням фінітних функцій та на основі метода зворотнього трасування.

Постановка завдання:

- Дослідження методів обробки зображень в системах візуалізації,
- Дослідження формування рельєфу з використанням фінітних функцій, на основі метода зворотнього трасування.
- Розробити модель на основі досліджених методів.

Модель геометричної обробки зображення на площині для растрових систем візуалізації



$$z = Z_0 - Y_0 \frac{c_{11}Z_э - c_{12}Y_э + c_{13}}{c_{31}Z_э - c_{32}Y_э + c_{33}},$$

$$x = X_0 - Y_0 \frac{c_{21}Z_э - c_{22}Y_э + c_{23}}{c_{31}Z_э - c_{32}Y_э + c_{33}},$$

Рисунок 1 - Система координат

Спостерігач методом зворотнього трасування

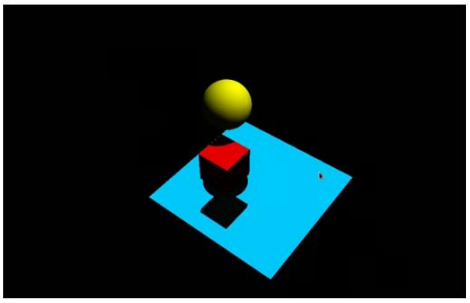


Рисунок 2 - Спостерігач

Фінітні функції

$$h_1 = h(x_ц, z_ц) \quad h_2 = h(x_ц, z_ц + 1) \quad h_3 = h(x_ц + 1, z_ц) \quad h_4 = h(x_ц + 1, z_ц + 1)$$

$$x_ц = \lfloor x \rfloor \quad x_д = x - x_ц \quad z_ц = \lfloor z \rfloor \quad z_д = z - z_ц$$

$$y(x,z) = h_1 f(x_д) f(z_д) + h_2 f(x_д)(1 - f(z_д)) + h_3 (1 - f(x_д)) f(z_д) + h_4 (1 - f(x_д))(1 - f(z_д)).$$

Блок схема алгоритму програми

Відображення рельєфу з використанням фінітних функцій

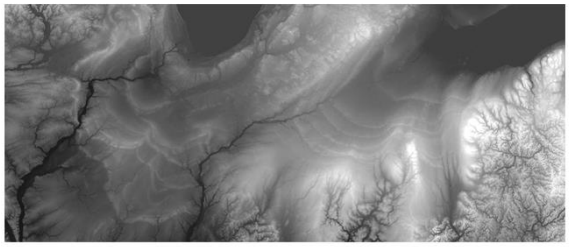


Рисунок 3 - Карта висот

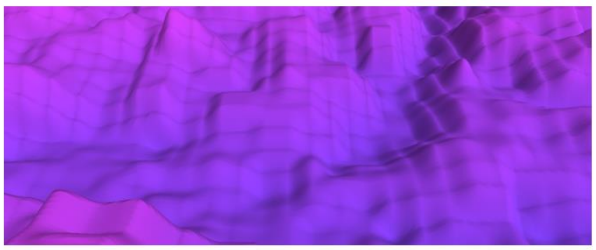


Рисунок 4 - Модель рельєфу

Деформації фінітних функцій у задачах синтезу зображення для систем візуалізації

Варіант 1

$$\tilde{y}(x,z)=\sum_{i=\hat{x}-1}^{\hat{x}+1}\sum_{j=\hat{z}-1}^{\hat{z}+1} y_{ij}\cdot f^1(x-i)\cdot f^1(z-j)=F^1(x_p)\hat{Y}^1\left(F^1(z_p)\right)^T$$

$$\hat{x}=\left\lfloor x+\tfrac{1}{2}\right\rfloor \quad \hat{z}=\left\lfloor z+\tfrac{1}{2}\right\rfloor \quad x_p=x-\hat{x},\ z_p=z-\hat{z},\quad x_p,z_p\in\left[-\tfrac{1}{2},\tfrac{1}{2}\right)$$

$F^1(t)=\left(f^1(t+1)\quad f^1(t)\quad f^1(t-1)\right)$ - вектор,

$\hat{Y}^1=\begin{pmatrix} y_{\hat{x}-1,\hat{z}-1} & y_{\hat{x}-1,\hat{z}} & y_{\hat{x}-1,\hat{z}+1} \\ y_{\hat{x},\hat{z}-1} & y_{\hat{x},\hat{z}} & y_{\hat{x},\hat{z}+1} \\ y_{\hat{x}+1,\hat{z}-1} & y_{\hat{x}+1,\hat{z}} & y_{\hat{x}+1,\hat{z}+1} \end{pmatrix}$ - матриця.

Рисунок 5 – Варіант 1

Варіант 2

$$\tilde{y}(x,z)=\sum_{i=x_ц}^{x_ц+1}\sum_{j=z_ц}^{z_ц+1} y_{ij}\cdot f^2(x-i)\cdot f^2(z-j)=F^2(x_д)\hat{Y}^2\left(F^2(z_д)\right)^T$$

$$x_ц=\lfloor x\rfloor \quad z_ц=\lfloor z\rfloor \quad x_д=x-x_ц \quad z_д=z-z_ц \quad x_д,z_д\in[0,1)$$

$F^2(t)=\left(f^2(t)\quad f^2(t-1)\right)$ - вектор,

$\hat{Y}^2=\begin{pmatrix} y_{x_ц,z_ц} & y_{x_ц,z_ц+1} \\ y_{x_ц+1,z_ц} & y_{x_ц+1,z_ц+1} \end{pmatrix}$ - матриця.

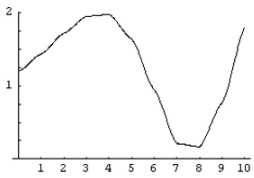


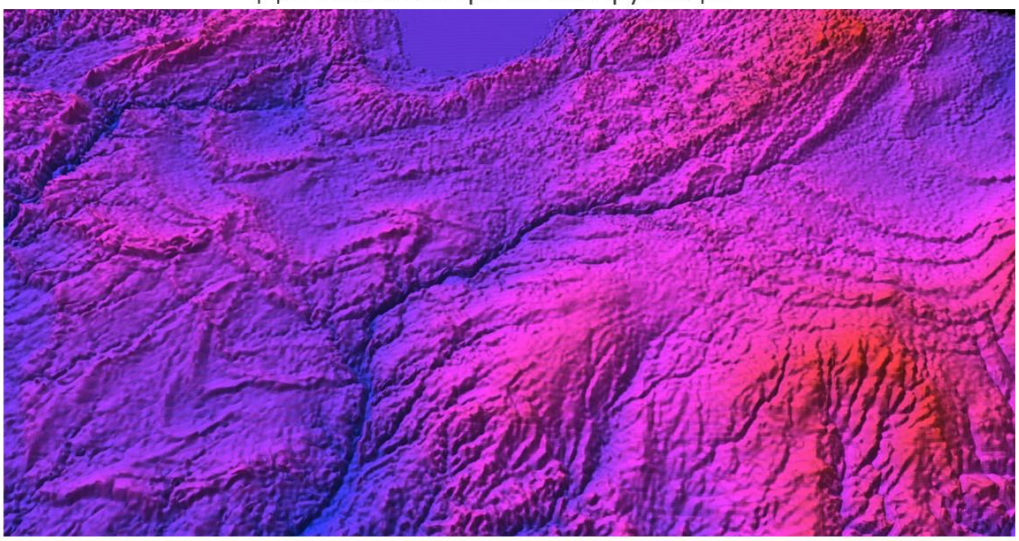
Рисунок 6 – Варіант 2

Відображення рельєфу з використанням метода деформування фінітних функцій

Рисунок 7 - Відображення рельєфу з використанням деформованих фінітних функцій

9

Модель рельєфу на основі метода зворотнього трасування та за допомогою фінітних функцій



10

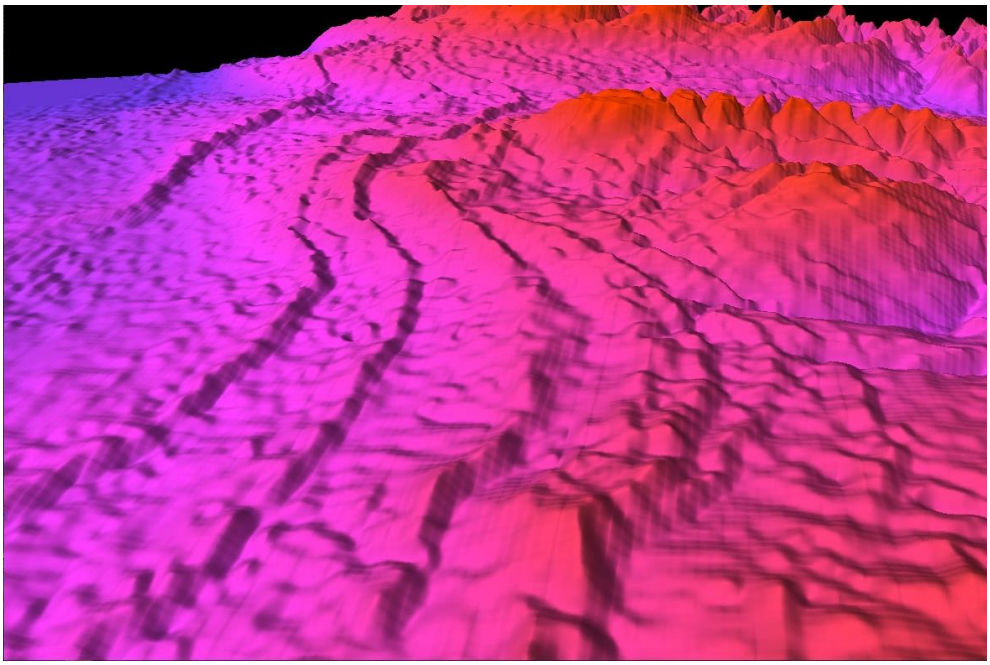
Висновки

В ході виконання атестаційної була побудована модель рельєфу в системах візуалізації. Досліджено що для синтезу зображень в реальному часі добре опрацьований метод прямого трасування. Докладно розглянутий метод зворотного трасування що дозволяє синтезувати зображення з високою реалістичністю. Досліджено рельєф в системах візуалізації для авіаційних тренажерів орієнтованих на метод зворотного трасування та за допомогою фінітних фукнцій. Модель показала що немає зламів на стиках суміжних поверхонь, побудова не вимагає попередніх обчислень, у базі даних зберігається мінімум інформації, що дозволяє компактно зберігати і швидко обробляти великі простори земної поверхні.

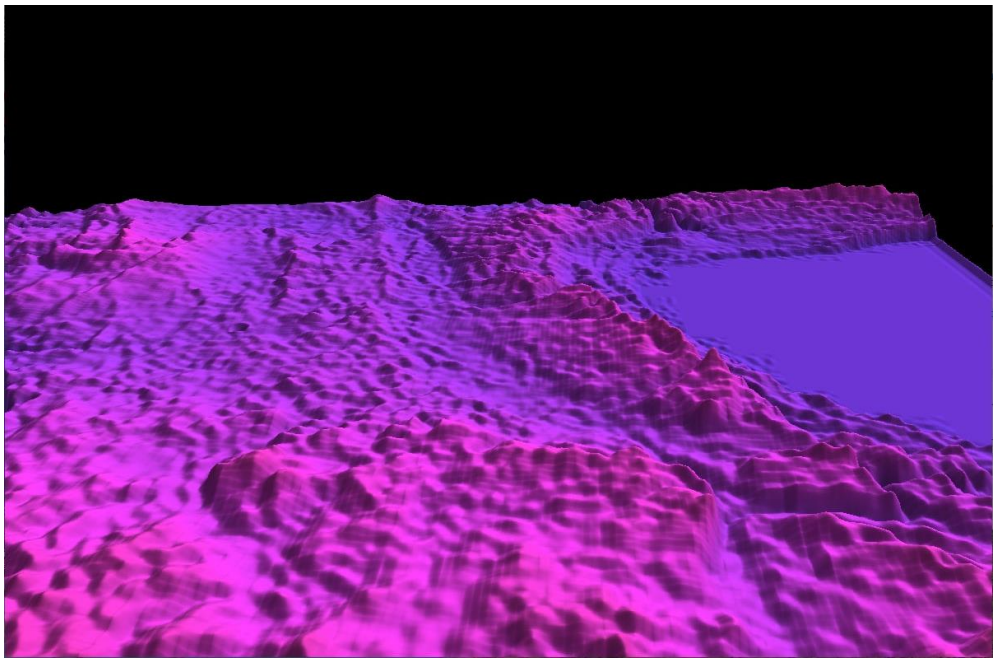
ДОДАТОК Б

Приклади роботи програми

Б.1 Гори на моделі рильєфу



Б.2 Низина на моделі рельєфу



ДОДАТОК В

Код програмного забезпечення

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <math.h>
#include <time.h>
#include <signal.h> //Сигнал, чтобы выходить по Ctrl+C, иначе SDL2 блокирует его
#include <string>
#include <SDL2/SDL.h>
#include <GL/glew.h>
#include <GL/gl.h>
#include <png.h>
typedef uint8_t u8;
typedef int8_t s8;
typedef uint16_t u16;
typedef int16_t s16;
typedef uint32_t u32;
typedef int32_t s32;
typedef uint64_t u64;
typedef int64_t s64;
typedef struct {
float x;
float y;
float z;
} vec3_t; //трёхмерный вектор

typedef struct {
float v[16];
/*
 * 0  4  8  12
 * 1  5  9  13
 * 2  6  10 14
 * 3  7  11 15
*/
} mat4_t; //матрица 4x4
mat4_t IdentityMatrix = {
{1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1},
}; //единичная матрица
vec3_t zerovec = {0};
//Некоторые математические операторы для векторов и матриц.
//Если тут матрица 4x4 умножается на трёхмерный вектор, то это означает, что вектор расширяется до четырёхмерного и последний элемент равен 1.0
vec3_t operator+(const vec3_t v0,const vec3_t v1) {
vec3_t out = v0;
out.x += v1.x;
out.y += v1.y;
out.z += v1.z;
return out;
}

vec3_t operator-(const vec3_t v0,const vec3_t v1) {
vec3_t out = v0;
out.x -= v1.x;
out.y -= v1.y;
```

```
out.z -= v1.z;
return out;
}
vec3_t operator*(const vec3_t v,float n) {
vec3_t out = v;
out.x *= n;
out.y *= n;
out.z *= n;
return out;
}
vec3_t operator*(const mat4_t mat,const vec3_t in) {
vec3_t out = {0};
out.x = in.x*mat.v[0] + in.y*mat.v[4] + in.z*mat.v[8] + 1*mat.v[12];
out.y = in.x*mat.v[1] + in.y*mat.v[5] + in.z*mat.v[9] + 1*mat.v[13];
out.z = in.x*mat.v[2] + in.y*mat.v[6] + in.z*mat.v[10] + 1*mat.v[14];
return out;
}
vec3_t operator-(const vec3_t in) {
vec3_t out = {0};
out.x = -in.x;
out.y = -in.y;
out.z = -in.z;
return out;
}
float dot(const vec3_t v0,const vec3_t v1) {
return (v0.x*v1.x + v0.y*v1.y + v0.z*v1.z);
}
float length(const vec3_t v) {
return sqrt(dot(v,v));
}
vec3_t cross(const vec3_t v0,const vec3_t v1) {
vec3_t ret;
ret.x = v0.y*v1.z - v0.z*v1.y;
ret.y = v0.z*v1.x - v0.x*v1.z;
ret.z = v0.x*v1.y - v0.y*v1.x;
return ret;
}
vec3_t normalize(const vec3_t v) {
float l = length(v);
vec3_t ret;
ret.x = v.x/l;
ret.y = v.y/l;
ret.z = v.z/l;
return ret;
}
float clamp(float x,float min,float max) {
if (x < min) {
      return min;
}
else if (x > max) {
      return max;
}
return x;
}
mat4_t operator*(const mat4_t m0,const mat4_t m1) {
mat4_t ret = {0};
s32 w = 4;
s32 h = 4;
s32 den = 4;
for (s32 y = 0; y < h; y++) {
      for (s32 x = 0; x < w; x++) {
            for (s32 i = 0; i < den; i++) {
                  ret.v[y*w + x] += m0.v[i*w + x]*m1.v[y*w + i];
            }
```

```
            }
        }
        return ret;
        }
        ////
        //Функции для преобразований трансформационной матрицы
        mat4_t TranslateMatrix(mat4_t dest,float x,float y,float z) {
        mat4_t tr = IdentityMatrix;
        tr.v[12] = x;
        tr.v[13] = y;
        tr.v[14] = z;
        return tr*dest;
        }
        mat4_t RotateMatrixX(mat4_t dest,float angle) {
        mat4_t rt = IdentityMatrix;
        rt.v[5] = cos(angle);
        rt.v[6] = sin(angle);
        rt.v[9] = -sin(angle);
        rt.v[10] = cos(angle);
        return rt*dest;
        }
        mat4_t RotateMatrixY(mat4_t dest,float angle) {
        mat4_t rt = IdentityMatrix;
        rt.v[0] = cos(angle);
        rt.v[2] = -sin(angle);
        rt.v[8] = sin(angle);
        rt.v[10] = cos(angle);
        return rt*dest;
        }
        mat4_t RotateMatrixZ(mat4_t dest,float angle) {
        mat4_t rt = IdentityMatrix;
        rt.v[0] = cos(angle);
        rt.v[1] = sin(angle);
        rt.v[4] = -sin(angle);
        rt.v[5] = cos(angle);
        return rt*dest;
        }
        mat4_t ScaleMatrix(mat4_t dest,float x,float y,float z) {
        mat4_t sc = IdentityMatrix;
        sc.v[0] = x;
        sc.v[5] = y;
        sc.v[10] = z;
        return sc*dest;
        }
        ////
        float DegToRad(float angle) { // Градусы в радианы
        return angle/360.0*(2*M_PI);
        }
        typedef struct {
        s32 w,h;
        float* heights; //массив высот размером w*h.
        } HeightMap_t; //Тип карты высот
        void dummyerror0() { //коллбек для libpng
        return;
        }
        void dummyerror1(struct png_struct_def * a, const char *b) { //коллбек
для libpng
        return;
        }
        void dummyerror2(struct png_struct_def * a, const char *b) { //коллбек
для libpng
        return;
        }
        u8* pngbytes;
```

```
	s32 pngsize;
	s32 pngseek;
	void readpngfn(png_structp png,png_bytep buf,png_size_t num) { //коллбек
для libpng
	if (pngseek + num < pngsize) {
		memcpy(buf,&pngbytes[pngseek],num);
		pngseek += num;
	}
	else {
		memcpy(buf,&pngbytes[pngseek],pngsize - pngseek);
		pngseek = pngsize - 1;
	}
	}
	//Функция загрузки height map из PNG-картинки
	HeightMap_t LoadHeightMap(std::string path) {
	s32 size;
	u8* data;
	FILE* f;
	f = fopen(path.c_str(),»rb»);
	if (f == NULL) {
		printf(«File %s does not exist\n»,path);
		exit(1);
	}
	fseek(f,0,SEEK_END);
	size = ftell(f);
	fseek(f,0,SEEK_SET);
	data = (u8*)malloc(size);
	fread(data,size,1,f);
	fclose(f);
	u32 width;
	u32 height;
	png_bytep* row_pointers;
	pngbytes = data;
	pngsize = size;
	pngseek = 0;
	png_structp                                  png                                  =
png_create_read_struct(PNG_LIBPNG_VER_STRING,(void*)dummyerror0,(void
(*)(png_struct_def  *,  const  char  *))dummyerror1,(void  (*)(png_struct_def  *,
const char *))dummyerror2); //Считать данные PNG структуры.
	png_set_read_fn(png,NULL,readpngfn);
	png_infop info = png_create_info_struct(png);
	png_read_info(png,info);
	width = png_get_image_width(png,info); //Получить ширину изображения
	height = png_get_image_height(png,info); //Получить высоту изображения
	u8 color_type = png_get_color_type(png,info);
	u8 bit_depth = png_get_bit_depth(png,info);
	//Тут  идут  разнообразные  вызовы  libpng,  которые  настроят  его  для
получения RGBA-значения.
	if (bit_depth == 16) {
		png_set_strip_16(png);
	}
	if (color_type == PNG_COLOR_TYPE_PALETTE) {
		png_set_palette_to_rgb(png);
	}
	if (color_type == PNG_COLOR_TYPE_GRAY && bit_depth < 8) {
		png_set_expand_gray_1_2_4_to_8(png);
	}
	if (png_get_valid(png,info,PNG_INFO_tRNS)) {
		png_set_tRNS_to_alpha(png);
	}
	if    (color_type    ==    PNG_COLOR_TYPE_RGB    ||    color_type    ==
PNG_COLOR_TYPE_GRAY || color_type == PNG_COLOR_TYPE_PALETTE) {
		png_set_filler(png,0xFF,PNG_FILLER_AFTER);
	}
```

```
	if (color_type == PNG_COLOR_TYPE_GRAY || color_type == PNG_COLOR_TYPE_GRAY_ALPHA) {
		png_set_gray_to_rgb(png);
	}
	png_read_update_info(png,info);
	row_pointers = (png_bytep*)malloc(sizeof(png_bytep)*height);
	for (int y = 0; y < height; y++) {
		row_pointers[y] = (png_byte*)malloc(png_get_rowbytes(png,info));
	}
	png_read_image(png,row_pointers);
	png_destroy_read_struct(&png,&info,NULL);
	png = NULL;
	info = NULL;
	HeightMap_t out;
	out.w = width;
	out.h = height;
	out.heights = (float*)malloc(sizeof(float)*out.w*out.h);
	for (u32 y = 0; y < height; y++) {
		for (u32 x = 0; x < width; x++) {
			out.heights[y*out.w + x] = row_pointers[y][x*4 + 0]/255.0; //Значени R пикселя
		}
	}
	for (int y = 0; y < height; y++) {
		free(row_pointers[y]);
	}
	free(row_pointers);
	free(data);
	return out;
	}

	SDL_Window* window; //окно
	s32 WindowWidth = 1200; // и его размеры
	s32 WindowHeight = 800;
	mat4_t camera = IdentityMatrix; //матрица камеры
	GLuint mainprogram; // Шейдер-программа опенгла
	HeightMap_t hm; //Основная карта высот
	GLuint hm_tex; //Opengl-текстура для карты высот
	void draw() {
	glLoadIdentity();
	glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT); //Очистить цвет и буфер глубины
	glClearColor(0.0,0.0,0.0,1.0);
	glBindTexture(GL_TEXTURE_2D,hm_tex);

	mat4_t camera_notrans = camera; //camera_notrans используется для вычисления направления векторов рейтрейсинга в фрагментном шейдере
	camera_notrans.v[12] = 0;
	camera_notrans.v[13] = 0;
	camera_notrans.v[14] = 0;
	GLfloat cam_pos[3] = {camera.v[12],camera.v[13],camera.v[14]};
	GLuint cam = glGetUniformLocation(mainprogram,»camera»);
	GLuint camnt = glGetUniformLocation(mainprogram,»camera_notrans»);
	GLuint camp = glGetUniformLocation(mainprogram,»camera_pos»);
	GLuint heighttex = glGetUniformLocation(mainprogram,»HeightMap»);
	GLuint heighttex_w = glGetUniformLocation(mainprogram,»HeightMapWidth»);
	GLuint heighttex_h = glGetUniformLocation(mainprogram,»HeightMapHeight»);
	glUseProgram(mainprogram);
	glUniformMatrix4fv(cam,1,0,camera.v); //Тут передаётся матрица камеры
	glUniformMatrix4fv(camnt,1,0,camera_notrans.v); //Тут передаётся матрица камеры с очищенным значением трансляции (перемещения). Нужно для образования направления векторов в шейдере.
```

```
	glUniform3f(camp,cam_pos[0],cam_pos[1],cam_pos[2]);   //Тут   передаётся
позиция камеры. Нужна для автоматической настройки разрешения в шейдере.
	glUniform1i(heighttex,GL_TEXTURE0);
	glUniform1i(heighttex_w,hm.w);
	glUniform1i(heighttex_h,hm.h);
	GLfloat vertex[8] = {-(float)WindowWidth/2, -(float)WindowHeight/2,
	                      (float)WindowWidth/2, -(float)WindowHeight/2,
	                      (float)WindowWidth/2,  (float)WindowHeight/2,
	                     -(float)WindowWidth/2,      (float)WindowHeight/2};
//рисовать прямоугольник во весь экран, иначе опенгл ничего не нарисует
	glVertexPointer(2,GL_FLOAT,0,vertex);
	glDrawArrays(GL_QUADS,0,4);
	glUseProgram(0);
	}
	void handler_sigint(s32 sig) { //выход по SIGINT сигналу (Ctrl+C)
	exit(0);
	}
	s32 oldmouse_x;
	s32 oldmouse_y;
	float angle_pitch;
	float angle_yaw;
	vec3_t pos;
	vec3_t mode1pos;
	float mode1r = 40.0;
	u8 disable = 1; //убрать курсор и выйти из режима полёта (для flymode =
0).
	u8 flymode = 1; //Используется только flymode = 1.
	float mousesens = 40.0; //чувствительность мыши
	#define FLYMODE1_KOEF 4.5
	void upr() {
	SDL_PumpEvents(); //Обновить SDL события.
	s32 mouse_x,mouse_y;
	s32  mousebuttons  =  SDL_GetMouseState(&mouse_x,&mouse_y);  //Получить
состояние мыши
	mouse_y  =  WindowHeight  -  mouse_y  -  1;  //Преобразовать  значение
координаты мыши для y, чтобы оно начинались не сверху окна, а снизу.
	s32 KeymapLen;
	const  Uint8*  keymap  =  SDL_GetKeyboardState(&KeymapLen);  //Узнать
состояние клавиш SDL-окна.
	if (flymode == 0) { //Полёт в режиме WASD+SPACE+F и управление камерой
мышью.
		if (disable) {
			s32 mouse_center_x = mouse_x - WindowWidth/2;
			s32 mouse_center_y = mouse_y - WindowHeight/2;

			angle_yaw -= (float)mouse_center_x/(50.001 - mousesens);
			angle_pitch += (float)mouse_center_y/(50.001 - mousesens);
		}
		if (disable) {
			SDL_WarpMouseInWindow(window,WindowWidth/2,WindowHeight/2);
			SDL_ShowCursor(SDL_DISABLE);
		}
		else {
			SDL_ShowCursor(SDL_ENABLE);
		}
	}
	else {
		s32  xval  =  mouse_x  -  oldmouse_x;  //тут  вычисляется  кол-во
пройденных курсором пикселей с прошлого кадра.
		s32 yval = mouse_y - oldmouse_y; ////
		if  (mousebuttons  &  (1  <<  0))  {  //Если  нажата  ЛКМ,  то  изменить
значение угла для расчёта камеры.
			angle_yaw          -=          (float)xval/(50.001          -
mousesens)*FLYMODE1_KOEF;
```

```
                angle_pitch        +=        (float)yval/(50.001        -
mousesens)*FLYMODE1_KOEF;
            }
            else if (mousebuttons & (1 << 1)) { //Если нажата средняя кнопка
мыши, то изменить mode1r, являющийся расстоянием от точки mode1pos до камеры.
                mode1r -= yval;
            }
            else if (mousebuttons & (1 << 2)) { //Если нажата ПКМ, то
изменить положение камеры.
                vec3_t                          in                          =
vec3_t{xval/(sqrt(mode1r)),yval/(sqrt(mode1r)),0}; //Тут вычисляется, как
далеко переместить её в зависимости от нынешнего mode1r. Это нужно, чтобы
скорость приблизительно соответствовала масштабу.

                mat4_t mat = IdentityMatrix;
                mat = RotateMatrixX(mat,DegToRad(angle_pitch));
                mat = RotateMatrixY(mat,DegToRad(angle_yaw));

                in  =  mat*in;  //Вектор  содержащий  значения  изменения
положения мыши трансформируется матрицей в соответствии с угловым положением
камеры.

                in  =  in*(pow(mode1r,0.7)/10.0);  //Тут  значение  ещё
уменьшается. 10.0 — просто коеффициент чувствительности.

                mode1pos.x -= in.x;
                mode1pos.y -= in.y;
                mode1pos.z -= in.z;
            }

            if (mode1r < 0.0) { //На случай, если пользователь приблизился
слишком близко, не дать mode1r стать отрицтательным.
                mode1r = 10;
            }
        }
        float yawrad = angle_yaw/(180.0/M_PI);
        if (keymap[SDL_SCANCODE_ESCAPE]) { //Выйти по Esc
            exit(0);
        }
        if   (keymap[SDL_SCANCODE_T])   {   //Для   flymode   =   0   изменить:
взаимодействовать с мышью или нет.
            disable = !disable;
        }
        //Управление для flymode = 0
        #define SPEED 5.0
        if (keymap[SDL_SCANCODE_SPACE]) {
            pos.y += SPEED;
        }
        if (keymap[SDL_SCANCODE_F]) {
            pos.y -= SPEED;
        }
        if (keymap[SDL_SCANCODE_W]) {
            pos.x -= SPEED*sin(yawrad);
            pos.z -= SPEED*cos(yawrad);
        }
        if (keymap[SDL_SCANCODE_S]) {
            pos.x += SPEED*sin(yawrad);
            pos.z += SPEED*cos(yawrad);
        }

        if (keymap[SDL_SCANCODE_D]) {
            pos.x += SPEED*cos(yawrad);
            pos.z -= SPEED*sin(yawrad);
        }
```

```
	if (keymap[SDL_SCANCODE_A]) {
		pos.x -= SPEED*cos(yawrad);
		pos.z += SPEED*sin(yawrad);
	}
	camera = IdentityMatrix;
	if (flymode == 0) { //Преобразование матрицы камеры для flymode = 0
		camera = RotateMatrixX(camera,DegToRad(angle_pitch));
		camera = RotateMatrixY(camera,DegToRad(angle_yaw));
		camera = TranslateMatrix(camera,pos.x,pos.y,pos.z);
	}
	else if (flymode == 1) { //Преобразование матрицы камеры для flymode =
1
		camera = TranslateMatrix(camera,0,0,mode1r);
		camera = RotateMatrixX(camera,DegToRad(angle_pitch));
		camera = RotateMatrixY(camera,DegToRad(angle_yaw));
		camera                                                    =
TranslateMatrix(camera,mode1pos.x,mode1pos.y,mode1pos.z);
	}
	oldmouse_x = mouse_x;
	oldmouse_y = mouse_y;
	}
	void InitGL(); //Объявление InitGL
	int  window_event_handler(void*  data,SDL_Event*  event)  {  //Функция
вызывается SDL2, чтобы обработать некоторые события
	if (event->type == SDL_WINDOWEVENT) {
		if (event->window.event == SDL_WINDOWEVENT_CLOSE) { //Если окно
должно быть закрыто, то выйти.
			exit(0);
		}
		else  if  (event->window.event  ==  SDL_WINDOWEVENT_RESIZED)  {
//Переинициализировать опенгл, когда окно ресайзнуто

			SDL_GetWindowSize(window,&WindowWidth,&WindowHeight);

			InitGL();
		}
	}
	else if (event->type == SDL_MOUSEWHEEL) { //Изменить mode1r для flymode
= 1 при прокрутке колёсиком мыши
		mode1r -= event->wheel.y*sqrt(mode1r);
	}

	return 0;
	}

	void PrintShaderError(GLuint shader,GLuint program) {
	char* msg;
	s32 InfoLogLength;

	if (program != 0) { //Узнать кол-во байт для вывода OpenGL-ошибки.
		glGetProgramiv(program,GL_INFO_LOG_LENGTH,&InfoLogLength);
	}
	else {
		glGetShaderiv(shader,GL_INFO_LOG_LENGTH,&InfoLogLength);
	}
	msg  =  (char*)calloc(1,InfoLogLength  +  1);  //Выделить  память  под
сообщение
	if (program != 0) { //Получить сообщение
		glGetProgramInfoLog(program,InfoLogLength,NULL,msg);
	}
	else {
		glGetShaderInfoLog(shader,InfoLogLength,NULL,msg);
	}
	printf(«shader error:\n»);
```

```
	printf(«%s\n»,msg); //Напечатать его
	free(msg); //Освободить память от него
	};
	float scale = 0.125;
	void InitGL() {
	glViewport(0,0,WindowWidth,WindowHeight); //Вьюпорт во весь экран
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	glOrtho(0.0,WindowWidth,0.0,WindowHeight,-1000000.0,1000000.0);
//Опенгл будет использовать ортогональную проекцию для отрисовки одного прямоугольника.
	glEnable(GL_TEXTURE_2D);
	glActiveTexture(GL_TEXTURE0);
	GLint success;
	FILE* f;
	char* data; //Указатель для данных, считанных с файла.
	s32 size; //Переменная для размера этих данных.

	if (mainprogram != 0) { //Для реинициализации
		glDeleteProgram(mainprogram);
	}
	mainprogram = glCreateProgram();
	f = fopen(«v.glsl»,»rb»); //Открыть файл вершинного шейдера
	fseek(f,0,SEEK_END); //Перевести курсор в конец файла
	size = ftell(f); //Узнать размер файла (нынешнее положение курсора)
	fseek(f,SEEK_SET,0); //Перевести курсор обратно
	data = (char*)malloc(size); //Выделить память для данных
	memset(data,0,size);
	fread(data,size,1,f);
	fclose(f);
	GLuint vertex_shader = glCreateShader(GL_VERTEX_SHADER); //Создать вершинный шейдер
	glShaderSource(vertex_shader,1,(const GLchar **)&data,&size);
//Загрузить код в него
	glCompileShader(vertex_shader); //Скомпилировать вершинный шейдер
	glGetShaderiv(vertex_shader,GL_COMPILE_STATUS,&success); //Получить ошибку, если есть.
	if (success != GL_TRUE) {
		PrintShaderError(vertex_shader,0);
	}
	glAttachShader(mainprogram,vertex_shader); //Добавить шейдер к программе mainprogram
	glLinkProgram(mainprogram); //Слинковать шейдерную программу

	glGetProgramiv(mainprogram,GL_LINK_STATUS,&success); //Узнать статус линковки.
	if (success != GL_TRUE) {
		PrintShaderError(0,mainprogram);
	}
	free(data);
	f = fopen(«fhm.glsl»,»rb»); //Открыть файл фрагментного шейдера и аналогично вышеописанному получить его размер и данные.
	fseek(f,0,SEEK_END);
	size = ftell(f);
	fseek(f,SEEK_SET,0);
	char* fdata = (char*)malloc(size + 1); //Но тут считать в переменную fdata
	memset(fdata,0,size + 1);
	fread(fdata,size,1,f);
	fclose(f);
	size += 200; //Добавить к размеру несколько байт, так как будет произведено форматирование через snprintf
	data = (char*)malloc(size);
	memset(data,0,size);
```

```
	snprintf(data,size,fdata,(float)WindowWidth,(float)WindowHeight,(float)
scale); //Форматирование. В шейдере есть текст %f, который заменяется на
значения размеров окна. Это нужно для рейтрейсинга.
	//printf(«%s\n»,fdata);
	//printf(«%s\n»,data);
	free(fdata);
	size = strlen((char*)data);
	GLuint frag_shader = glCreateShader(GL_FRAGMENT_SHADER); //Создать
фрагментный шейдер
	glShaderSource(frag_shader,1,(const GLchar **)&data,&size);
//Аналогично загрузить код и скомпилировать его
	glCompileShader(frag_shader);
	glGetShaderiv(frag_shader,GL_COMPILE_STATUS,&success);
	if (success != GL_TRUE) {
		PrintShaderError(frag_shader,0);
	}
	glAttachShader(mainprogram,frag_shader);
	glBindAttribLocation(mainprogram,0,»in_Position»); //Указать, что
переменная вершинного шейдера in_Position имеет нулевой номер.
	glLinkProgram(mainprogram);
	glGetProgramiv(mainprogram,GL_LINK_STATUS,&success);
	if (success != GL_TRUE) {
		PrintShaderError(0,mainprogram);
	}
	free(data);
	}
	std::string filename = «a.png»;
	int main(int argc,char* argv[]) {
	signal(SIGINT,&handler_sigint); //Пробросить обработку SIGINT на
handler_sigint
	for (s32 i = 1; i < argc; i++) {
		if (!strcmp(argv[i],(char*)»--scale»)) {
			if (i + 1 < argc) {
				i++;
				float sc = atof(argv[i]);
				if (sc > 0.0) {
					scale = sc;
				}
				else {
					printf(«Invalid scale: %f\n»,sc);
					exit(1);
				}
			}
		}
		else {
			filename = argv[i];
		}
	}

	if (SDL_Init(SDL_INIT_EVERYTHING) != 0) { //Инициализировать шейдер. В
случае ошибки, напечать об этом и выйти.
		printf(«SDL_Init failed: %s\n», SDL_GetError());
		return 1;
	}
	//Инициализация SDL-окна.
	SDL_DisplayMode dispmode;
	SDL_GetCurrentDisplayMode(0,&dispmode);
	window =
SDL_CreateWindow(««,200,200,WindowWidth,WindowHeight,SDL_WINDOW_RESIZABLE |
SDL_WINDOW_OPENGL);
	SDL_AddEventWatch(window_event_handler,window); //Добавить функцию
window_event_handler для обработки событий
	//Создать и инициализировать контекст опенгла
	SDL_GLContext glcontext = SDL_GL_CreateContext(window);
```

```
	SDL_GL_SetAttribute(SDL_GL_CONTEXT_PROFILE_MASK,SDL_GL_CONTEXT_PROFILE_
CORE);
	SDL_GL_SetAttribute(SDL_GL_CONTEXT_MAJOR_VERSION,3);
	SDL_GL_SetAttribute(SDL_GL_CONTEXT_MINOR_VERSION,2);
	SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER,1);
	GLenum error = glewInit();//Инициализировать GLEW, в случае ошибки
напечать об этом и выйти.
	if (error != GLEW_OK) {
		printf(«glewInit failed: %s\n»,glewGetErrorString(error));
		return 1;
	}
	SDL_GL_SetSwapInterval(1);
	hm = LoadHeightMap(filename);
	InitGL(); //Инициализировать опенгл и шейдеры.
	glGenTextures(1,&hm_tex);
	glBindTexture(GL_TEXTURE_2D,hm_tex);
	glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
	glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
	glTexImage2D(GL_TEXTURE_2D,0,1,hm.w,hm.h,0,GL_RED,GL_FLOAT,hm.heights);
	angle_pitch = -30; //Поставить изначальный угол камеры в -30 градусов.
		struct timespec time1,time2; //структуры для измерения
пройденного времени.
	glEnableClientState(GL_VERTEX_ARRAY); //Включить GL_VERTEX_ARRAY для
рисовки через glVertexPointer и glDrawArrays
	while (1) {
		clock_gettime(CLOCK_MONOTONIC,&time1); //Получить время до
рисовки
		draw();
		SDL_GL_SwapWindow(window); //Свапнуть кадр в окне.
		upr(); //Управление
		clock_gettime(CLOCK_MONOTONIC,&time2); //Получить время после
рисовки
		u64 usec = time2.tv_sec*1000000 + time2.tv_nsec/1000 -
time1.tv_sec*1000000 - time1.tv_nsec/1000;
		printf(«FPS: %f\n»,1000000.0/usec); //Вывести нынешний FPS
	}



	uniform mat4 camera;
	uniform mat4 camera_notrans;
	uniform vec3 camera_pos;
	uniform int HeightMapWidth;
	uniform int HeightMapHeight;
	uniform sampler2D HeightMap;
	out vec4 FragColor; //Выходной цвет
	#define WIDTH %f //Ширина окна
	#define HEIGHT %f //Высота окна
	#define SCALE %f
	#define FOV 1.309 //Угол обзора в радинах
	vec3 objposs[1] = {vec3(0.0,0.0,0.0)};
	vec3 colors[1] = {vec3(1.0,1.0,1.0)}; //Цвета объектов
	vec3 lightposs[1] = {vec3(0.0,100.0,0.0)}; //Положение источника света.
	struct CastRet_t { //Структура данных о песечение луча.
	bool casted; //Произошло ли пересечение

	vec3 normal; //Нормаль поверхности, где произошло пересечение
	vec3 intersect; //Позиция точки пересечения
	};
	#define M_PI 3.14159265358979323846264338327795
	float finit_cos(float t) { //Финитная фукнция.
	return (cos(M_PI*t) + 1.0)/2.0*int(t < 1.0);
	}
	float finit_cos_deform(float t) { //Деформированная финитная функция
```

```
	return 2.0/3.0*finit_cos(t*2.0/3.0);
	}
	float HeightByCoord(float x,float z) { //Получить высоту по координатам
	float h = 0;
	int sx = int(x + 0.5);
	int sz = int(z + 0.5);
	for (int ii = -1; ii <= 1; ii++) {
		for (int jj = -1; jj <= 1; jj++) {
			int i = sx + ii;
			int j = sz + jj;
			float tuth =
texture(HeightMap,vec2(float(i)/HeightMapWidth,float(j)/HeightMapHeight)).r*2
55.0*SCALE;
			h += tuth*finit_cos_deform(x - i)*finit_cos_deform(z - j);
		}
	}
	return h;
	}
	#define NORMAL_EPSILON 0.004
	vec3 NormalByCoord(float x,float z) { //Получить нормаль по координатам.
Тут используется приближённое значение градиента путём вызова функции
HeightByCoord с добавлением или отнятием NORMAL_EPSILON
	float hx = HeightByCoord(x + NORMAL_EPSILON,z) - HeightByCoord(x -
NORMAL_EPSILON,z);
	float hz = HeightByCoord(x,z + NORMAL_EPSILON) - HeightByCoord(x,z -
NORMAL_EPSILON);

	return normalize(vec3(hx,NORMAL_EPSILON/2,hz)); //y-вая компонента
вектора можно выбрать на глаз, влияет на «высоту» источника освещения.
	}
	#define DT 0.2
	#define MAXT_PLUS 100.0
	#define MAX_ITERATIONS 2000
	CastRet_t TraceObjects(vec3 pos,vec3 dir) { //Трассировка объектов.
	CastRet_t maincast; //Основная переменная для сохранения результатов
пересечения.
	maincast.casted = false;
	float maxt = camera_pos.y + HeightMapWidth + HeightMapHeight +
MAXT_PLUS; //Максимальное расстояние для точки, перемещающейся по лучу
	float dt = DT; //Величина, на которую будет перемещаться точка луча в
каждой итерации
	if (camera_pos.y > 80.0) {
		dt *= pow(camera_pos.y,0.5);
	}
	else if (camera_pos.y > 300.0) {
		dt *= pow(camera_pos.y,0.3);
	}
	if (maxt/dt > MAX_ITERATIONS) { //Защита от перегрузки.
		maxt = maxt/(maxt/dt)*(MAX_ITERATIONS - 10);
	}
	for (float t = 0; t < maxt; t += dt) {
		vec3 nowpos = pos + dir*t;
		if ((dir.y < 0.0 && nowpos.y < -dt) || (dir.y > 0.0 && nowpos.y >
255.0*SCALE)) { //Если луч опустился ниже карты высот или выше её максимально
возможного значения, то выйти из цикла
			break;
		}
		float h = HeightByCoord(nowpos.x,nowpos.z);
		if (nowpos.y < h && (nowpos.x >= 0.0 && nowpos.x < HeightMapWidth
&& nowpos.z >= 0.0 && nowpos.z < HeightMapHeight)) { //Если луч попадает на
карту высот и точка ниже нынешней высоты, то выйти из цикла с записью данных
о пересечении
			maincast.intersect = nowpos;
			maincast.casted = true;
```

```
                maincast.normal = NormalByCoord(nowpos.x,nowpos.z);
                break;
            }
        }

        return maincast;
        }
        #define MINCOLOR 0.1 //Минимальное значение цвета, чтобы объекты не
могли быть совсем чёрными и были видны даже в тени.
        vec3 RaytraceObjects(vec3 pos,vec3 dir) { //Трассировка объектов, теней
и получение итогового RGB-значения цвета для пикселя.
        vec3 color = vec3(0.0,0.0,0.0);
        CastRet_t maincast = TraceObjects(pos,dir);
        if (maincast.casted) { //Если произошло пересечение, значит дальнейшая
обработка
            float yval = (maincast.intersect.y/(255.0*SCALE))*2.0;
            color      =      vec3(clamp(yval,0.0,1.0),0.25,clamp(2.0      -
yval,0.0,1.0))/1.2; //Цвет варьируется от синего к красному, в завимисоти от
значения высоты. Делится на 1.2, чтобы не слишком ярко было.
            color                =                color*0.7                +
color*0.3*dot(maincast.normal,vec3(1.0,1.0,1.0));  //Тут  только  0.3  части
цвета поддаётся затенения.
        }
        return color;
        }
        void main() { //Эта функция вызывается опенглом для каждого пикселя
окна.
        //Получения значений положения пикселя окна.
        float x = gl_FragCoord.x;
        float y = gl_FragCoord.y;

        vec3 pos = (camera*vec4(0.0,0.0,0.0,1.0)).xyz; //Позиция луча
        vec3       dir       =       (camera_notrans*normalize(vec4((x       -
WIDTH/2)/WIDTH*tan(FOV/2)*WIDTH/HEIGHT,(y    -    HEIGHT/2)/HEIGHT*tan(FOV/2),-
1.0,1.0))).xyz; //Направление луча
        FragColor  =  vec4(RaytraceObjects(pos,dir),1.0);  //Получить  итоговый
цвет для пикселя
        }

        //Просто шейдер, чтобы положения вертексов остались теми же, что и есть.

        in vec3 in_Position;

        void main(void) {
        gl_Position = mat4(1.0)*vec4(in_Position,1.0);
        }
```