Sasu Alexandru-Cristian
Grupa 342

Scenă în deșert

I. Conceptul proiectului

Este prezentată o scenă dintr-un deșert, ce are ca inspirație desenul animat american “Courage the Cowardly Dog”. Astfel, pe o suprafață 2D, căreia îi este aplicată o textură de pământ crăpat, este dispusă o casă 3D cu geamuri și ușă, alături de o turbină eoliană 3D a cărei elice se învârte cu o viteză constantă. De asemenea, este presentă și o sursă de lumină, care acționează ca un soare, generând umbre pentru casă și turbina eoliană.

II. Elementele incluse

- O multitudine de obiecte 3D: casa, geamurile casei, ușa casei, corpul turbinei eoliene.
- O textură, aplicată suprafeței pe care sunt dispuse obiectele, căreia i s-a aplicat procedeul de normal mapping.
- Sursă de lumină, cu cei trei termeni specifici: ambiental, difuz, specular.
- Umbre pentru obiectele din scenă, generate in funcție de caracteristicile sursei de lumină, cât și de caracteristicile obiectelor.
- Un observator care survolează scena (prin apăsarea tastelor săgeți, cât și prin apăsarea semnelor - și +), deplasându-se pe o sferă invizibilă.

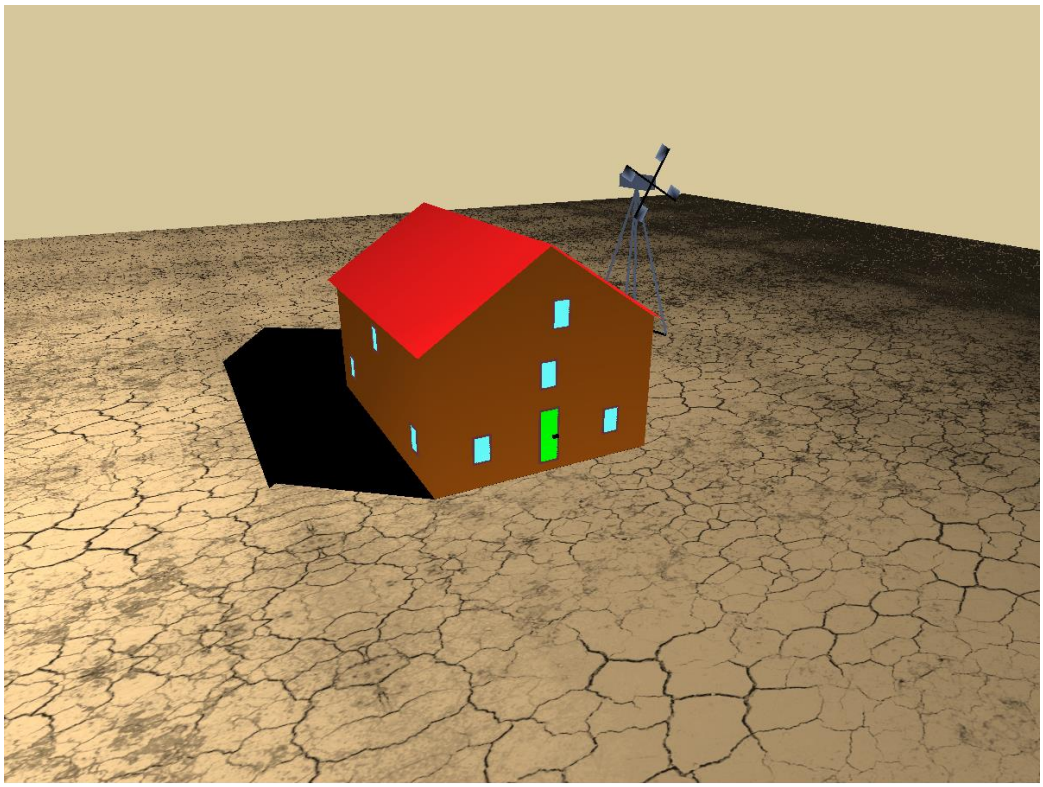
III. Originalitatea proiectului

Proiectul este original deoarece își dorește să prezinte scena desfășurării acțiunii din desenul animat american “Courage the Cowardly Dog”.

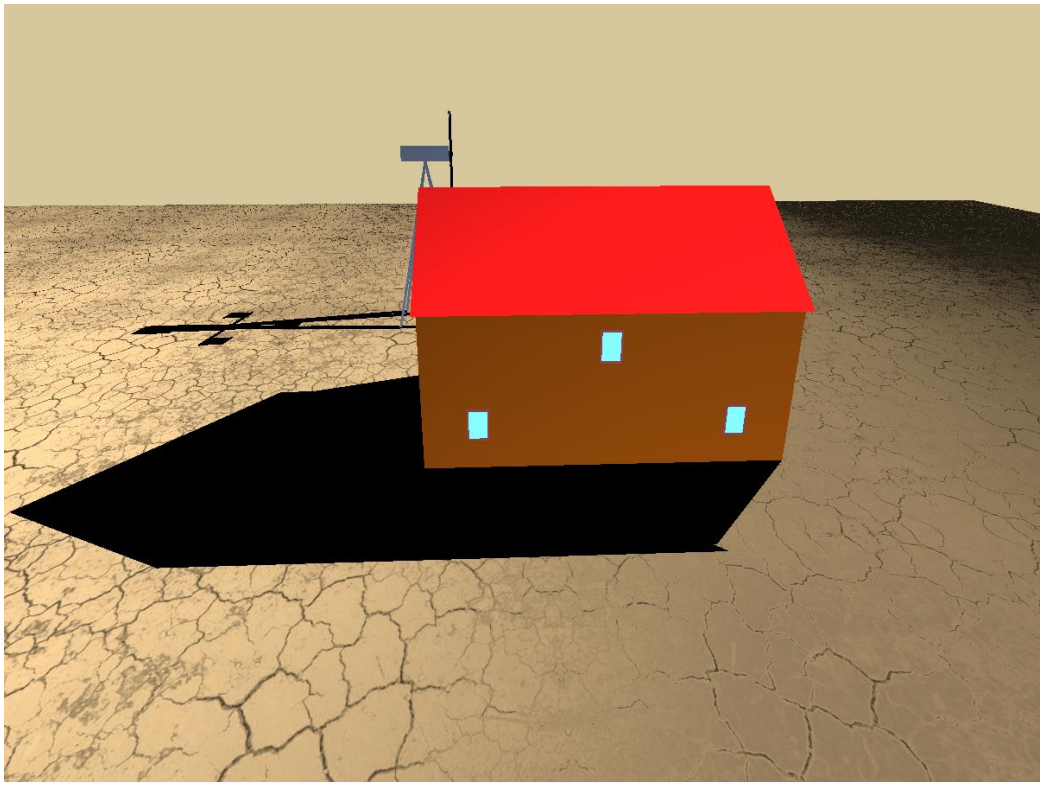
Sasu Alexandru-Cristian
Grupa 342

IV. Prezentarea proiectului

Fața scenei:

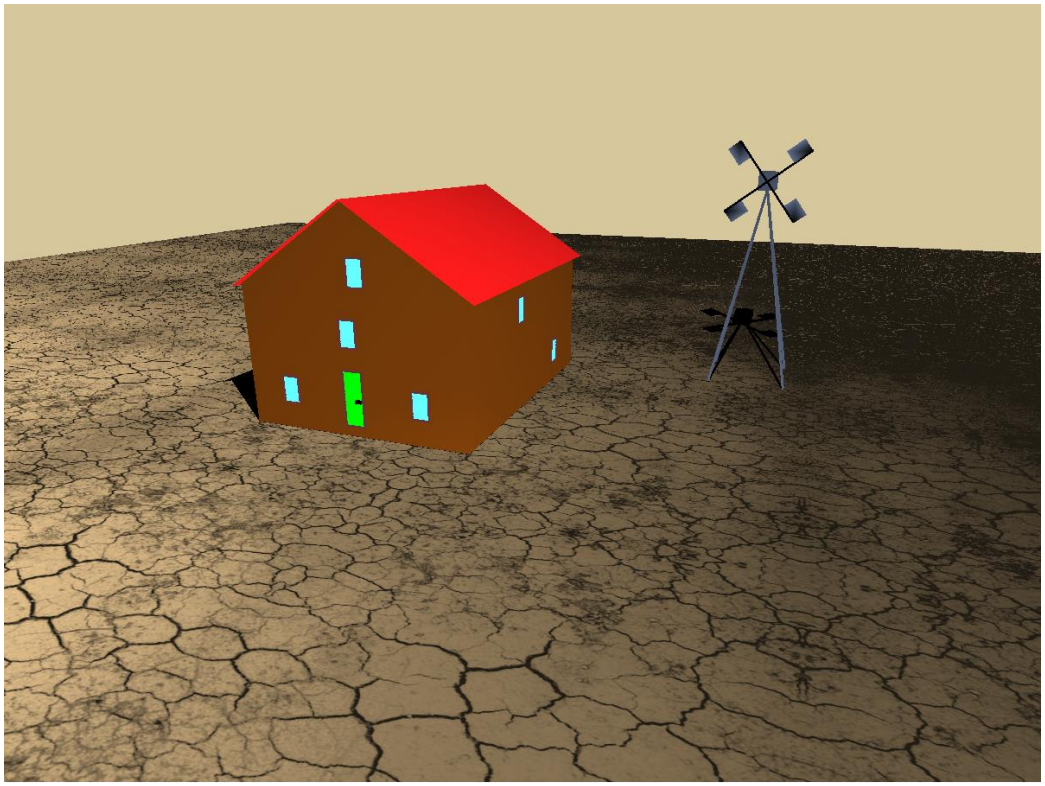


Lateralul scenei (stânga):

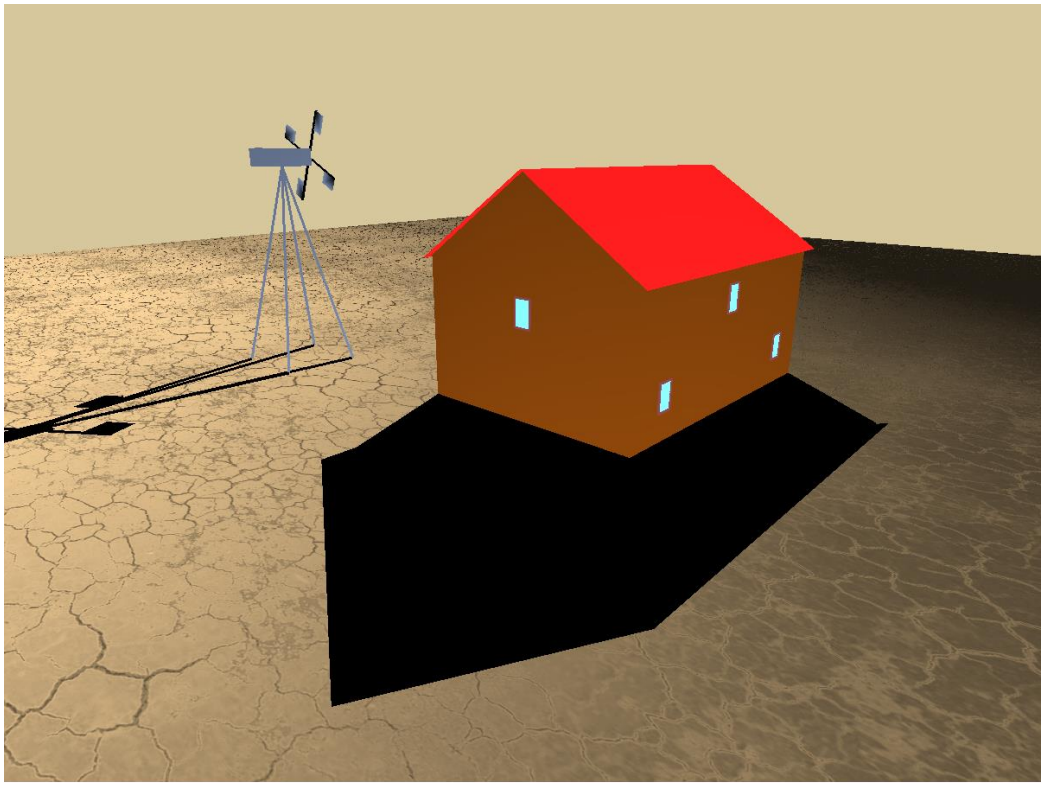


Sasu Alexandru-Cristian
Grupa 342

Lateralul scenei (dreapta):



Spatele scenei:



Sasu Alexandru-Cristian
Grupa 342

Porțiunea de cod care încarcă texturile:

```
void LoadTexture(int texType)
{
    if (texType == 0)
    {
        glGenTextures(1, &texture);
        glBindTexture(GL_TEXTURE_2D, texture);
    }
    else
        if (texType == 1)
        {
            glGenTextures(1, &normalMap);
            glBindTexture(GL_TEXTURE_2D, normalMap);
        }

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT);

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

    int width, height, nrChannels;
    if (texType == 0)
    {
        unsigned char* image = stbi_load("CrackedGroundTexture.jpg", &width, &height, &nrChannels, 0);
        if (image)
        {
            glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, image);
            glGenerateMipmap(GL_TEXTURE_2D);
            stbi_image_free(image);
        }
        else
        {
            cout << "Failed to load texture" << endl;
        }
    }
    else
        if (texType == 1)
        {
            unsigned char* image = stbi_load("NormalMap.jpg", &width, &height, &nrChannels, 0);
            if (image)
            {
                glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, image);
                glGenerateMipmap(GL_TEXTURE_2D);
                stbi_image_free(image);
            }
            else
            {
                cout << "Failed to load normal map" << endl;
            }
        }
}
```

Sasu Alexandru-Cristian
Grupa 342

Resursele utilizate:

https://learnopengl.com/Getting-started/Textures

https://raw.githubusercontent.com/nothings/stb/master/stb_image.h

https://learnopengl.com/Advanced-Lighting/Normal-Mapping

https://www.youtube.com/watch?v=JNj1A1bl7gg&ab_channel=VictorGordan

https://cpetry.github.io/NormalMap-Online/

Cod din laboratoarele trecute

Sasu Alexandru-Cristian
Grupa 342

Anexe

Codul sursă:

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <GL/glew.h>
#include <GL/freeglut.h>
#include "loadShaders.h"
#include "glm/glm.hpp"
#include "glm/gtc/matrix_transform.hpp"
#include "glm/gtx/transform.hpp"
#include "glm/gtc/type_ptr.hpp"

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"

using namespace std;

// Identifiers
GLuint
VaoIdGround, VboIdGround, EboIdGround,
VaoIdHouse, VboIdHouse, EboIdHouse,
VaoIdDoor, VboIdDoor, EboIdDoor,
VaoIdWindow, VboIdWindow, EboIdWindow,
VaoIdTurbine, VboIdTurbine, EboIdTurbine,
VaoIdPropeller, VboIdPropeller, EboIdPropeller,
ColorBufferId,
```

Sasu Alexandru-Cristian
Grupa 342

```
ProgramId,

myMatrixLocation,

matShadowLocation,

viewLocation,

projLocation,

matrRotlLocation,

lightColorLocation,

lightPosLocation,

viewPosLocation,

colorCodeLocation,

textureId = 0, normalMapId = 1;


// Helper variables

int colorCode;

float const PI = 3.141592f;


// Elements for the view matrix

float Refx = 0.0f, Refy = 0.0f, Refz = 0.0f;

float alpha = PI / 8, beta = 0.0f, dist = 3000.0f;

float Obsx, Obsy, Obsz;

float Vx = 0.0, Vy = 0.0, Vz = 1.0;

glm::mat4 view;


// Elements for the projection matrix

float width = 800, height = 600, znear = 0.1, fov = 45;

glm::mat4 projection;


// Light source coordinates

float xL = 2000.0f, yL = 1000.0f, zL = 2400.0f;


// The shadow matrix

float matShadow[4][4];
```

Sasu Alexandru-Cristian
Grupa 342

```
// Matrices used for modeling transformations

glm::mat4

myMatrix,

matTranHouse,

matTranDoor,

matRotWindow,

matTranWindowFront1, matTranWindowFront2, matTranWindowFront3, matTranWindowFront4,

matTranWindowBack,

matTranWindowRightSide1, matTranWindowRightSide2,

matTranWindowLeftSide1, matTranWindowLeftSide2, matTranWindowLeftSide3,

matTranTurbine, matTranPropeller, matRotPropeller;


// Elements for the house

float

houseTranX = -300.0f, houseTranY = -400.0f, houseTranZ = 0.0f,

doorXRelativeToHouseTranXIncrement = 905.0f, doorYRelativeToHouseTranYIncrement = 120.0f,
doorZRelativeToHouseTranZIncrement = -15.0f,

windowFront1XRelativeToHouseTranXIncrement = 905.0f, windowFront1YRelativeToHouseTranYIncrement =
-100.0f, windowFront1ZRelativeToHouseTranZIncrement = 35.0f,

windowFront2XRelativeToHouseTranXIncrement = 905.0f, windowFront2YRelativeToHouseTranYIncrement =
350.0f, windowFront2ZRelativeToHouseTranZIncrement = 35.0f,

windowFront3XRelativeToHouseTranXIncrement = 905.0f, windowFront3YRelativeToHouseTranYIncrement =
120.0f, windowFront3ZRelativeToHouseTranZIncrement = 230.0f,

windowFront4XRelativeToHouseTranXIncrement = 905.0f, windowFront4YRelativeToHouseTranYIncrement =
160.0f, windowFront4ZRelativeToHouseTranZIncrement = 410.0f,

windowBackXRelativeToHouseTranXIncrement = -110.0f, windowBackYRelativeToHouseTranYIncrement =
120.0f, windowBackZRelativeToHouseTranZIncrement = 230.0f,

windowRightSide1XRelativeToHouseTranXIncrement = 600.0f,
windowRightSide1YRelativeToHouseTranYIncrement = 610.0f,
windowRightSide1ZRelativeToHouseTranZIncrement = 250.0f,

windowRightSide2XRelativeToHouseTranXIncrement = 250.0f,
windowRightSide2YRelativeToHouseTranYIncrement = 610.0f,
windowRightSide2ZRelativeToHouseTranZIncrement = 25.0f,

windowLeftSide1XRelativeToHouseTranXIncrement = 850.0f,
windowLeftSide1YRelativeToHouseTranYIncrement = -150.0f,
windowLeftSide1ZRelativeToHouseTranZIncrement = 35.0f,

windowLeftSide2XRelativeToHouseTranXIncrement = 500.0f,
windowLeftSide2YRelativeToHouseTranYIncrement = -150.0f,
windowLeftSide2ZRelativeToHouseTranZIncrement = 250.0f,
```

```
windowLeftSide3XRelativeToHouseTranXIncrement = 150.0f,
windowLeftSide3YRelativeToHouseTranYIncrement = -150.0f,
windowLeftSide3ZRelativeToHouseTranZIncrement = 35.0f;



// Elements for the wind turbine

float

angle = 0, rotationSpeed = 0.2,

turbineTranX = 0.0f, turbineTranY = 800.0f, turbineTranZ = 0.0f,

propellerTranXRelativeToTurbineTranXIncrement = -265.0f,
propellerTranYRelativeToTurbineTranYIncrement = 0.0f,
propellerTranZRelativeToTurbineTranZIncrement = 780.0f;



void generateTransformationMatrices(void)

{

	matTranHouse = glm::translate(glm::mat4(1.0f), glm::vec3(houseTranX, houseTranY,
houseTranZ));

	matRotWindow = glm::rotate(glm::mat4(1.0f), -PI / 2, glm::vec3(0.0, 0.0, 1.0));

	matTranDoor = glm::translate(glm::mat4(1.0f), glm::vec3(houseTranX +
doorXRelativeToHouseTranXIncrement, houseTranY + doorYRelativeToHouseTranYIncrement, houseTranZ +
doorZRelativeToHouseTranZIncrement));

	matTranWindowFront1 = glm::translate(glm::mat4(1.0f), glm::vec3(houseTranX +
windowFront1XRelativeToHouseTranXIncrement, houseTranY +
windowFront1YRelativeToHouseTranYIncrement, houseTranZ +
windowFront1ZRelativeToHouseTranZIncrement));

	matTranWindowFront2 = glm::translate(glm::mat4(1.0f), glm::vec3(houseTranX +
windowFront2XRelativeToHouseTranXIncrement, houseTranY +
windowFront2YRelativeToHouseTranYIncrement, houseTranZ +
windowFront2ZRelativeToHouseTranZIncrement));

	matTranWindowFront3 = glm::translate(glm::mat4(1.0f), glm::vec3(houseTranX +
windowFront3XRelativeToHouseTranXIncrement, houseTranY +
windowFront3YRelativeToHouseTranYIncrement, houseTranZ +
windowFront3ZRelativeToHouseTranZIncrement));

	matTranWindowFront4 = glm::translate(glm::mat4(1.0f), glm::vec3(houseTranX +
windowFront4XRelativeToHouseTranXIncrement, houseTranY +
windowFront4YRelativeToHouseTranYIncrement, houseTranZ +
windowFront4ZRelativeToHouseTranZIncrement));

	matTranWindowBack = glm::translate(glm::mat4(1.0f), glm::vec3(houseTranX +
windowBackXRelativeToHouseTranXIncrement, houseTranY + windowBackYRelativeToHouseTranYIncrement,
houseTranZ + windowBackZRelativeToHouseTranZIncrement));

	matTranWindowRightSide1 = glm::translate(glm::mat4(1.0f), glm::vec3(houseTranX +
windowRightSide1XRelativeToHouseTranXIncrement, houseTranY +
windowRightSide1YRelativeToHouseTranYIncrement, houseTranZ +
windowRightSide1ZRelativeToHouseTranZIncrement));

	matTranWindowRightSide2 = glm::translate(glm::mat4(1.0f), glm::vec3(houseTranX +
windowRightSide2XRelativeToHouseTranXIncrement, houseTranY +
```

```
windowRightSide2YRelativeToHouseTranYIncrement, houseTranZ +
windowRightSide2ZRelativeToHouseTranZIncrement));

	matTranWindowLeftSide1 = glm::translate(glm::mat4(1.0f), glm::vec3(houseTranX +
windowLeftSide1XRelativeToHouseTranXIncrement, houseTranY +
windowLeftSide1YRelativeToHouseTranYIncrement, houseTranZ +
windowLeftSide1ZRelativeToHouseTranZIncrement));

	matTranWindowLeftSide2 = glm::translate(glm::mat4(1.0f), glm::vec3(houseTranX +
windowLeftSide2XRelativeToHouseTranXIncrement, houseTranY +
windowLeftSide2YRelativeToHouseTranYIncrement, houseTranZ +
windowLeftSide2ZRelativeToHouseTranZIncrement));

	matTranWindowLeftSide3 = glm::translate(glm::mat4(1.0f), glm::vec3(houseTranX +
windowLeftSide3XRelativeToHouseTranXIncrement, houseTranY +
windowLeftSide3YRelativeToHouseTranYIncrement, houseTranZ +
windowLeftSide3ZRelativeToHouseTranZIncrement));



	matTranTurbine = glm::translate(glm::mat4(1.0f), glm::vec3(turbineTranX, turbineTranY,
turbineTranZ));

	matTranPropeller = glm::translate(glm::mat4(1.0f), glm::vec3(turbineTranX +
propellerTranXRelativeToTurbineTranXIncrement, turbineTranY +
propellerTranYRelativeToTurbineTranYIncrement, turbineTranZ +
propellerTranZRelativeToTurbineTranZIncrement));

	angle += rotationSpeed;

	matRotPropeller = glm::rotate(glm::mat4(1.0f), angle, glm::vec3(1.0, 0.0, 0.0));

}



void processNormalKeys(unsigned char key, int x, int y)

{

	switch (key)

	{

	case 'l':

		Vx -= 0.1;

		break;

	case 'r':

		Vx += 0.1;

		break;

	case '+':

		dist += 35;

		break;

	case '-':

		dist -= 35;
```

```
		break;
	}
	if (key == 27)
		exit(0);
}
void processSpecialKeys(int key, int xx, int yy)
{
	switch (key)
	{
	case GLUT_KEY_LEFT:
		beta -= 0.06;
		break;
	case GLUT_KEY_RIGHT:
		beta += 0.06;
		break;
	case GLUT_KEY_UP:
		alpha += 0.06;
		break;
	case GLUT_KEY_DOWN:
		alpha -= 0.06;
		break;
	}
}


void CreateVBOGround(void)
{
	GLfloat Vertices[] =
	{
	    // Positions                                    // Colors               // Normals
// Texture coords
	    -5000.0f, -5000.0f, 40.0f, 1.0f,   0.65f, 0.55f, 0.4f,   0.0f, 0.0f, 1.0f,   0.0f,
0.0f,
		 5000.0f, -5000.0f, 40.0f, 1.0f,   0.65f, 0.55f, 0.4f,   0.0f, 0.0f, 1.0f,   10.0f,
0.0f,
```

```
		5000.0f,  5000.0f, 40.0f, 1.0f,   0.65f, 0.55f, 0.4f,   0.0f, 0.0f, 1.0f,   10.0f, 10.0f,
	-5000.0f,  5000.0f, 40.0f, 1.0f,   0.65f, 0.55f, 0.4f,   0.0f, 0.0f, 1.0f,   0.0f, 10.0f
	};

	GLubyte Indices[] =
	{
		0, 1, 2,   0, 2, 3,
	};

	glGenVertexArrays(1, &VaoIdGround);
	glGenBuffers(1, &VboIdGround);
	glGenBuffers(1, &EboIdGround);
	glBindVertexArray(VaoIdGround);

	glBindBuffer(GL_ARRAY_BUFFER, VboIdGround);
	glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);
	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboIdGround);
	glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indices, GL_STATIC_DRAW);

	// Attribute 0 = position
	glEnableVertexAttribArray(0);
	glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 12 * sizeof(GLfloat), (GLvoid*)0);
	// Attribute 1 = color
	glEnableVertexAttribArray(1);
	glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 12 * sizeof(GLfloat), (GLvoid*)(4 * sizeof(GLfloat)));
	// Attribute 2 = normal
	glEnableVertexAttribArray(2);
	glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 12 * sizeof(GLfloat), (GLvoid*)(7 * sizeof(GLfloat)));
	// Attribute 3 = texture coord
	glEnableVertexAttribArray(3);
```

```
	glVertexAttribPointer(3, 2, GL_FLOAT, GL_FALSE, 12 * sizeof(GLfloat), (GLvoid*)(10 * sizeof(GLfloat)));

}

void CreateVBOHouse(void)

{

	GLfloat Vertices[] =

	{

		// Positions                              // Colors                // Normals

		0.0f,    -250.0f, 51.0f,  1.0f,   0.4f, 0.2f, 0.03f,   0.0f, 0.0f, 1.0f,  // Vertex 0

		0.0f,     500.0f, 51.0f,  1.0f,   0.4f, 0.2f, 0.03f,   0.0f, 0.0f, 1.0f,

		0.0f,     500.0f, 500.0f, 1.0f,   0.4f, 0.2f, 0.03f,   0.0f, 0.0f, 1.0f,

		0.0f,    -250.0f, 500.0f, 1.0f,   0.4f, 0.2f, 0.03f,   0.0f, 0.0f, 1.0f,

		0.0f,     125.0f, 750.0f, 1.0f,   0.4f, 0.2f, 0.03f,   0.0f, 0.0f, 1.0f,


		1000.0f, -250.0f, 51.0f,  1.0f,   0.4f, 0.2f, 0.03f,   0.0f, 0.0f, 1.0f,  // Vertex 5

		1000.0f,  500.0f, 51.0f,  1.0f,   0.4f, 0.2f, 0.03f,   0.0f, 0.0f, 1.0f,

		1000.0f,  500.0f, 500.0f, 1.0f,   0.4f, 0.2f, 0.03f,   0.0f, 0.0f, 1.0f,

		1000.0f, -250.0f, 500.0f, 1.0f,   0.4f, 0.2f, 0.03f,   0.0f, 0.0f, 1.0f,

		1000.0f,  125.0f, 750.0f, 1.0f,   0.4f, 0.2f, 0.03f,   0.0f, 0.0f, 1.0f,


	    -10.0f,   -280.0f, 490.0f, 1.0f,   0.9f, 0.1f, 0.1f,    0.0f, 0.0f, 1.0f,  // Vertex 10

		-10.0f,    125.0f, 755.0f, 1.0f,   0.9f, 0.1f, 0.1f,    0.0f, 0.0f, 1.0f,

		-10.0f,    530.0f, 490.0f, 1.0f,   0.9f, 0.1f, 0.1f,    0.0f, 0.0f, 1.0f,

		 1010.0f,  530.0f, 490.0f, 1.0f,   0.9f, 0.1f, 0.1f,    0.0f, 0.0f, 1.0f,

		 1010.0f,  125.0f, 755.0f, 1.0f,   0.9f, 0.1f, 0.1f,    0.0f, 0.0f, 1.0f,

		 1010.0f, -280.0f, 490.0f, 1.0f,   0.9f, 0.1f, 0.1f,    0.0f, 0.0f, 1.0f

	};


	GLubyte Indices[] =

	{

		  0,  1,  3,    3,  1,  2,    2,  3,  4,                    // Front of the house

		  5,  6,  8,    8,  6,  7,    7,  8,  9,                    // Back of the house
```

```
            0,  1,  5,    1,  5,  6,                                          // Base of the
house

            1,  2,  6,    2,  6,  7,                                          // Right side of
the house

            2,  4,  7,    4,  7,  9,                                          // Upper right
side of the house

            0,  3,  5,    3,  5,  8,                                          // Left side of
the house

            3,  4,  8,    4,  8,  9,                                          // Upper left
side of the house

            11, 12, 13,   11, 13, 14,   10, 11, 15,   11, 14, 15  // Roof of the house

      };


      glGenVertexArrays(1, &VaoIdHouse);

      glGenBuffers(1, &VboIdHouse);

      glGenBuffers(1, &EboIdHouse);

      glBindVertexArray(VaoIdHouse);


      glBindBuffer(GL_ARRAY_BUFFER, VboIdHouse);

      glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);

      glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboIdHouse);

      glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indices, GL_STATIC_DRAW);


      // Attribute 0 = position

      glEnableVertexAttribArray(0);

      glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)0);

      // Attribute 1 = color

      glEnableVertexAttribArray(1);

      glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)(4 *
sizeof(GLfloat)));

      // Attribute 2 = normal

      glEnableVertexAttribArray(2);

      glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)(7 *
sizeof(GLfloat)));

}

void CreateVBODoor(void)
```

Sasu Alexandru-Cristian
Grupa 342

```
{
	GLfloat Vertices[] =
	{
		// Positions				// Colors			// Normals
		100.0f, -30.0f, 100.0f,  1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,  // Vertex 0
		100.0f,  40.0f, 100.0f,  1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,
		100.0f,  40.0f, 280.0f,  1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,
		100.0f, -30.0f, 280.0f,  1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,

		105.0f, -30.0f, 100.0f,  1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,  // Vertex 4
		105.0f,  40.0f, 100.0f,  1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,
		105.0f,  40.0f, 280.0f,  1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,
		105.0f, -30.0f, 280.0f,  1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,

		100.0f, -25.0f, 105.0f,  1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,  // Vertex 8
		100.0f,  35.0f, 105.0f,  1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,
		100.0f,  35.0f, 275.0f,  1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,
		100.0f, -25.0f, 275.0f,  1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,

		105.0f, -25.0f, 105.0f,  1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,  // Vertex
12
		105.0f,  35.0f, 105.0f,  1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,
		105.0f,  35.0f, 275.0f,  1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,
		105.0f, -25.0f, 275.0f,  1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,

		102.5f, -25.0f, 105.0f,  1.0f,   0.0f, 1.0f, 0.0f,    0.0f, 0.0f, 1.0f,  // Vertex
16
		102.5f,  35.0f, 105.0f,  1.0f,   0.0f, 1.0f, 0.0f,    0.0f, 0.0f, 1.0f,
		102.5f,  35.0f, 275.0f,  1.0f,   0.0f, 1.0f, 0.0f,    0.0f, 0.0f, 1.0f,
		102.5f, -25.0f, 275.0f,  1.0f,   0.0f, 1.0f, 0.0f,    0.0f, 0.0f, 1.0f,

		102.5f,  15.0f, 180.0f,  1.0f,   0.0f, 0.0f, 0.0f,    0.0f, 0.0f, 1.0f,  // Vertex
20
		102.5f,  35.0f, 180.0f,  1.0f,   0.0f, 0.0f, 0.0f,    0.0f, 0.0f, 1.0f,
```

```
            102.5f,  35.0f, 190.0f,  1.0f,   0.0f, 0.0f, 0.0f,    0.0f, 0.0f, 1.0f,

            102.5f,  15.0f, 190.0f,  1.0f,   0.0f, 0.0f, 0.0f,    0.0f, 0.0f, 1.0f,


            112.5f,  15.0f, 180.0f,  1.0f,   0.0f, 0.0f, 0.0f,    0.0f, 0.0f, 1.0f,  // Vertex
24
            112.5f,  35.0f, 180.0f,  1.0f,   0.0f, 0.0f, 0.0f,    0.0f, 0.0f, 1.0f,

            112.5f,  35.0f, 190.0f,  1.0f,   0.0f, 0.0f, 0.0f,    0.0f, 0.0f, 1.0f,

            112.5f,  15.0f, 190.0f,  1.0f,   0.0f, 0.0f, 0.0f,    0.0f, 0.0f, 1.0f,

      };


      GLubyte Indices[] =

      {

            0,  1,  9,    0,  8,  9,    1,  2,  9,    2,  9,  10,   2,  10, 11,   2,  3,  11,
3,  0,  11,   0,  8,  11,                                          // Front of the door frame

            4,  5,  13,   4,  12, 13,   5,  6,  13,   6,  13, 14,   6,  14, 15,   6,  7,  15,
7,  4,  15,   4,  12, 15,                                          // Back of the door frame

            0,  1,  4,    1,  4,  5,    1,  2,  5,    2,  5,  6,    2,  3,  6,    3,  6,  7,
3,  0,  4,    3,  4,  7,                                           // Outer sides of the door
frame

            8,  9,  12,   9,  12, 13,   9,  10, 13,   10, 13, 14,   10, 11, 14,   11, 14, 15,
11, 8,  12,   11, 12, 15,                                          // Inner sides of the door
frame

            16, 17, 18,   16, 18, 19,

                    // The door

            24, 25, 26,   24, 26, 27,   20, 21, 24,   21, 24, 25,   22, 23, 26,   23, 26, 27,
21, 22, 25,   22, 25, 26,   20, 23, 24,   23, 24, 27  // The doorknob

      };


      glGenVertexArrays(1, &VaoIdDoor);

      glGenBuffers(1, &VboIdDoor);

      glGenBuffers(1, &EboIdDoor);

      glBindVertexArray(VaoIdDoor);


      glBindBuffer(GL_ARRAY_BUFFER, VboIdDoor);

      glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);

      glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboIdDoor);

      glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indices, GL_STATIC_DRAW);
```

```
	// Attribute 0 = position
	glEnableVertexAttribArray(0);
	glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)0);
	// Attribute 1 = color
	glEnableVertexAttribArray(1);
	glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)(4 *
sizeof(GLfloat)));
	// Attribute 2 = normal
	glEnableVertexAttribArray(2);
	glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)(7 *
sizeof(GLfloat)));
}
void CreateVBOWindow(void)
{
	GLfloat Vertices[] =
	{
		// Positions				// Colors			// Normals
		100.0f, -30.0f, 100.0f, 1.0f,	0.4f, 0.2f, 0.23f,	0.0f, 0.0f, 1.0f,  // Vertex 0
		100.0f,  30.0f, 100.0f, 1.0f,	0.4f, 0.2f, 0.23f,	0.0f, 0.0f, 1.0f,
		100.0f,  30.0f, 190.0f, 1.0f,	0.4f, 0.2f, 0.23f,	0.0f, 0.0f, 1.0f,
		100.0f, -30.0f, 190.0f, 1.0f,	0.4f, 0.2f, 0.23f,	0.0f, 0.0f, 1.0f,

		105.0f, -30.0f, 100.0f, 1.0f,	0.4f, 0.2f, 0.23f,	0.0f, 0.0f, 1.0f,  // Vertex 4
		105.0f,  30.0f, 100.0f, 1.0f,	0.4f, 0.2f, 0.23f,	0.0f, 0.0f, 1.0f,
		105.0f,  30.0f, 190.0f, 1.0f,	0.4f, 0.2f, 0.23f,	0.0f, 0.0f, 1.0f,
		105.0f, -30.0f, 190.0f, 1.0f,	0.4f, 0.2f, 0.23f,	0.0f, 0.0f, 1.0f,

		100.0f, -25.0f, 105.0f, 1.0f,	0.4f, 0.2f, 0.23f,	0.0f, 0.0f, 1.0f,  // Vertex 8
		100.0f,  25.0f, 105.0f, 1.0f,	0.4f, 0.2f, 0.23f,	0.0f, 0.0f, 1.0f,
		100.0f,  25.0f, 185.0f, 1.0f,	0.4f, 0.2f, 0.23f,	0.0f, 0.0f, 1.0f,
		100.0f, -25.0f, 185.0f, 1.0f,	0.4f, 0.2f, 0.23f,	0.0f, 0.0f, 1.0f,

		105.0f, -25.0f, 105.0f, 1.0f,	0.4f, 0.2f, 0.23f,	0.0f, 0.0f, 1.0f,  // Vertex 12
```

```
            105.0f,  25.0f, 105.0f, 1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,

            105.0f,  25.0f, 185.0f, 1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,

            105.0f, -25.0f, 185.0f, 1.0f,   0.4f, 0.2f, 0.23f,   0.0f, 0.0f, 1.0f,


            102.5f, -25.0f, 105.0f, 1.0f,   0.4f, 1.0f, 1.0f,    0.0f, 0.0f, 1.0f,  // Vertex 16

            102.5f,  25.0f, 105.0f, 1.0f,   0.4f, 1.0f, 1.0f,    0.0f, 0.0f, 1.0f,

            102.5f,  25.0f, 185.0f, 1.0f,   0.4f, 1.0f, 1.0f,    0.0f, 0.0f, 1.0f,

            102.5f, -25.0f, 185.0f, 1.0f,   0.4f, 1.0f, 1.0f,    0.0f, 0.0f, 1.0f

      };


      GLubyte Indices[] =

      {

            0,  1,  9,    0,  8,  9,    1, 2,  9,    2,  9,  10,   2,  10, 11,   2,  3,  11,
3,  0, 11,   0,  8,  11,  // Front of the window frame

            4,  5,  13,   4,  12, 13,   5, 6,  13,   6,  13, 14,   6,  14, 15,   6,  7,  15,
7,  4, 15,   4,  12, 15,  // Back of the window frame

            0,  1,  4,    1,  4,  5,    1, 2,  5,    2,  5,  6,    2,  3,  6,    3,  6,  7,
3,  0, 4,    3,  4,  7,   // Outer sides of the window frame

            8,  9,  12,   9,  12, 13,   9, 10, 13,   10, 13, 14,   10, 11, 14,   11, 14, 15,
11, 8, 12,   11, 12, 15,  // Inner sides of the window frame

            16, 17, 18,   16, 18, 19
                                                                    // The window

      };


      glGenVertexArrays(1, &VaoIdWindow);

      glGenBuffers(1, &VboIdWindow);

      glGenBuffers(1, &EboIdWindow);

      glBindVertexArray(VaoIdWindow);


      glBindBuffer(GL_ARRAY_BUFFER, VboIdWindow);

      glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);

      glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboIdWindow);

      glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indices, GL_STATIC_DRAW);


      // Attribute 0 = position
```

```
	glEnableVertexAttribArray(0);

	glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)0);

	// Attribute 1 = color

	glEnableVertexAttribArray(1);

	glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)(4 *
sizeof(GLfloat)));

	// Attribute 2 = normal

	glEnableVertexAttribArray(2);

	glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)(7 *
sizeof(GLfloat)));

}

void CreateVBOWindTurbine(void)

{

	GLfloat Vertices[] =

	{

		// Positions                                    // Colors                    //
Normals

		-510.0f, -135.0f, 51.0f,  1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,  // Vertex
0

		-510.0f,  135.0f, 51.0f,  1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,

		-240.0f,  135.0f, 51.0f,  1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,

		-240.0f, -135.0f, 51.0f,  1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,

		-375.0f,  0.0f,   750.0f, 1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,


		-475.0f, -30.0f,  750.0f, 1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,  // Vertex
5

		-475.0f,  30.0f,  750.0f, 1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,

		-275.0f,  30.0f,  750.0f, 1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,

		-275.0f, -30.0f,  750.0f, 1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,

		-475.0f, -30.0f,  810.0f, 1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,

		-475.0f,  30.0f,  810.0f, 1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,

		-275.0f,  30.0f,  810.0f, 1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,

		-275.0f, -30.0f,  810.0f, 1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f

	};
```

```
	GLubyte Indices[] =

	{

		0, 4,   1, 4,   2, 4,   3, 4,  // Legs of the wind turbine

		// Motor of the wind turbine

		5, 6,  7,    5, 7,  8,

		9, 10, 11,   9, 11, 12,

		6, 7,  11,   6, 11, 10,

		8, 5,  9,    8, 9,  12,

		5, 6,  10,   5, 10, 9,

		7, 8,  12,   7, 12, 11

	};


	glGenVertexArrays(1, &VaoIdTurbine);

	glGenBuffers(1, &VboIdTurbine);

	glGenBuffers(1, &EboIdTurbine);

	glBindVertexArray(VaoIdTurbine);


	glBindBuffer(GL_ARRAY_BUFFER, VboIdTurbine);

	glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);

	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboIdTurbine);

	glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indices, GL_STATIC_DRAW);


	// Attribute 0 = position

	glEnableVertexAttribArray(0);

	glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)0);

	// Attribute 1 = color

	glEnableVertexAttribArray(1);

	glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)(4 *
sizeof(GLfloat)));

	// Attribute 2 = normal

	glEnableVertexAttribArray(2);

	glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)(7 *
sizeof(GLfloat)));

}
```

Sasu Alexandru-Cristian
Grupa 342

```
void CreateVBOPropeller(void)

{

	GLfloat Vertices[] =

	{

		// Positions                          // Colors                    // Normals

		0.0f, -168.75f,  0.0f,    1.0f,   0.0f,  0.0f, 0.0f,   0.0f, 0.0f, 1.0f,  // Vertex
0

		0.0f, -101.25f,  0.0f,    1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,

		0.0f, -101.25f, -47.5f,   1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,

		0.0f, -168.75f, -47.5f,   1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,


		0.0f,  168.75f,  0.0f,    1.0f,   0.0f,  0.0f, 0.0f,   0.0f, 0.0f, 1.0f,  // Vertex
4

		0.0f,  101.25f,  0.0f,    1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,

		0.0f,  101.25f,  47.5f,   1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,

		0.0f,  168.75f,  47.5f,   1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,


		0.0f,  0.0f,    -168.75f, 1.0f,   0.0f,  0.0f, 0.0f,   0.0f, 0.0f, 1.0f,  // Vertex
8

		0.0f,  0.0f,    -101.25f, 1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,

		0.0f,  47.5f,   -101.25f, 1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,

		0.0f,  47.5f,   -168.75f, 1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,


		0.0f,  0.0f,     168.75f, 1.0f,   0.0f,  0.0f, 0.0f,   0.0f, 0.0f, 1.0f,  // Vertex
12

		0.0f,  0.0f,     101.25f, 1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,

		0.0f, -47.5f,    101.25f, 1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f,

		0.0f, -47.5f,    168.75f, 1.0f,   0.35f, 0.4f, 0.5f,   0.0f, 0.0f, 1.0f

	};


	GLubyte Indices[] =

	{

		0, 4,   8, 12,  // Propeller skeleton

		// Rest of the propeller

		0,  1,  2,    0,  2,  3,
```

```
		4,  5,  6,    4,  6,  7,
		8,  9,  10,   8,  10, 11,
		12, 13, 14,   12, 14, 15
	};


	glGenVertexArrays(1, &VaoIdPropeller);
	glGenBuffers(1, &VboIdPropeller);
	glGenBuffers(1, &EboIdPropeller);
	glBindVertexArray(VaoIdPropeller);


	glBindBuffer(GL_ARRAY_BUFFER, VboIdPropeller);
	glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);
	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboIdPropeller);
	glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indices, GL_STATIC_DRAW);


	// Attribute 0 = position
	glEnableVertexAttribArray(0);
	glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)0);
	// Attribute 1 = color
	glEnableVertexAttribArray(1);
	glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)(4 *
sizeof(GLfloat)));
	// Attribute 2 = normal
	glEnableVertexAttribArray(2);
	glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(GLfloat), (GLvoid*)(7 *
sizeof(GLfloat)));
}
void DestroyVBO(void)
{
	glDisableVertexAttribArray(3);
	glDisableVertexAttribArray(2);
	glDisableVertexAttribArray(1);
	glDisableVertexAttribArray(0);
	glBindBuffer(GL_ARRAY_BUFFER, 0);
```

```
	glDeleteBuffers(1, &VboIdGround);
	glDeleteBuffers(1, &EboIdGround);
	glDeleteBuffers(1, &VboIdHouse);
	glDeleteBuffers(1, &EboIdHouse);
	glDeleteBuffers(1, &VboIdDoor);
	glDeleteBuffers(1, &EboIdDoor);
	glDeleteBuffers(1, &VboIdWindow);
	glDeleteBuffers(1, &EboIdWindow);
	glDeleteBuffers(1, &VboIdTurbine);
	glDeleteBuffers(1, &EboIdTurbine);
	glDeleteBuffers(1, &VboIdPropeller);
	glDeleteBuffers(1, &EboIdPropeller);
	glBindVertexArray(0);
	glDeleteVertexArrays(1, &VaoIdGround);
	glDeleteVertexArrays(1, &VaoIdHouse);
	glDeleteVertexArrays(1, &VaoIdDoor);
	glDeleteVertexArrays(1, &VaoIdWindow);
	glDeleteVertexArrays(1, &VaoIdTurbine);
	glDeleteVertexArrays(1, &VaoIdPropeller);
}

void LoadTexture(int texType)
{
	if (texType == 0)
	{
		glGenTextures(1, &textureId);
		glBindTexture(GL_TEXTURE_2D, textureId);
	}
	else
		if (texType == 1)
		{
			glGenTextures(1, &normalMapId);
			glBindTexture(GL_TEXTURE_2D, normalMapId);
```

```
		}


	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);

	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT);


	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);


	int width, height, nrChannels;

	if (texType == 0)

	{

		unsigned char* image = stbi_load("CrackedGroundTexture.jpg", &width, &height,
&nrChannels, 0);

		if (image)

		{

			glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
GL_UNSIGNED_BYTE, image);

			glGenerateMipmap(GL_TEXTURE_2D);

			stbi_image_free(image);

		}

		else

		{

			cout << "Failed to load texture" << endl;

		}

	}

	else

		if (texType == 1)

		{

			unsigned char* image = stbi_load("NormalMap.jpg", &width, &height,
&nrChannels, 0);

			if (image)

			{

				glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
GL_UNSIGNED_BYTE, image);

				glGenerateMipmap(GL_TEXTURE_2D);
```

```
                        stbi_image_free(image);
                }
                else
                {
                        cout << "Failed to load normal map" << endl;
                }
        }
}

void CreateShaders(void)
{
        ProgramId = LoadShaders("proiect_2_Shader.vert", "proiect_2_Shader.frag");
        glUseProgram(ProgramId);
}

void DestroyShaders(void)
{
        glDeleteProgram(ProgramId);
}

void Initialize(void)
{
        myMatrix = glm::mat4(1.0f);

        glClearColor(0.84f, 0.78f, 0.61f, 0.0f);

        CreateVBOGround();
        CreateVBOHouse();
        CreateVBODoor();
        CreateVBOWindow();
        CreateVBOWindTurbine();
        CreateVBOPropeller();
        CreateShaders();
```

```
	// Variable locations for the shaders
	myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
	matShadowLocation = glGetUniformLocation(ProgramId, "matShadow");
	viewLocation = glGetUniformLocation(ProgramId, "view");
	projLocation = glGetUniformLocation(ProgramId, "projection");
	lightColorLocation = glGetUniformLocation(ProgramId, "lightColor");
	lightPosLocation = glGetUniformLocation(ProgramId, "lightPos");
	viewPosLocation = glGetUniformLocation(ProgramId, "viewPos");
	colorCodeLocation = glGetUniformLocation(ProgramId, "colorCode");
}
void RenderFunction(void)
{
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glEnable(GL_DEPTH_TEST);

	generateTransformationMatrices();

	// Observer position
	Obsx = Refx + dist * cos(alpha) * cos(beta);
	Obsy = Refy + dist * cos(alpha) * sin(beta);
	Obsz = Refz + dist * sin(alpha);

	// Visualization + projection matrices
	glm::vec3 Obs = glm::vec3(Obsx, Obsy, Obsz);    // Observer position changes
	glm::vec3 RefPt = glm::vec3(Refx, Refy, Refz);  // Reference point position
	glm::vec3 Vert = glm::vec3(Vx, Vy, Vz);                 // Vertical line from field of
view
	view = glm::lookAt(Obs, RefPt, Vert);
	glUniformMatrix4fv(viewLocation, 1, GL_FALSE, &view[0][0]);
	projection = glm::infinitePerspective(fov, GLfloat(width) / GLfloat(height), znear);
	glUniformMatrix4fv(projLocation, 1, GL_FALSE, &projection[0][0]);
```

```
	// Shadow matrix

	float D = -50.0f;

	matShadow[0][0] = zL + D; matShadow[0][1] = 0; matShadow[0][2] = 0; matShadow[0][3] = 0;

	matShadow[1][0] = 0; matShadow[1][1] = zL + D; matShadow[1][2] = 0; matShadow[1][3] = 0;

	matShadow[2][0] = -xL; matShadow[2][1] = -yL; matShadow[2][2] = D; matShadow[2][3] = -1;

	matShadow[3][0] = -D * xL; matShadow[3][1] = -D * yL; matShadow[3][2] = -D * zL;
matShadow[3][3] = zL;

	glUniformMatrix4fv(matShadowLocation, 1, GL_FALSE, &matShadow[0][0]);


	// Uniform variables for illumination

	glUniform3f(lightColorLocation, 1.0f, 1.0f, 1.0f);

	glUniform3f(lightPosLocation, xL, yL, zL);

	glUniform3f(viewPosLocation, Obsx, Obsy, Obsz);


	// Preparing the textures for drawing the ground

	int texType = 0;

	LoadTexture(texType);

	glActiveTexture(GL_TEXTURE0 + 0);

	glBindTexture(GL_TEXTURE_2D, textureId);

	texType = 1;

	LoadTexture(texType);

	glActiveTexture(GL_TEXTURE0 + 1);

	glBindTexture(GL_TEXTURE_2D, normalMapId);


	// Drawing the ground

	glBindVertexArray(VaoIdGround);

	colorCode = 0;

	glUniform1i(colorCodeLocation, colorCode);

	myMatrix = glm::mat4(1.0f);

	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);

	glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, (void*)(0));

	colorCode = -1;

	glUniform1i(colorCodeLocation, colorCode);
```

```
	glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0);
	glUniform1i(glGetUniformLocation(ProgramId, "myNormalMap"), 1);
	glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, (void*)(0));

	// Drawing the house
	glBindVertexArray(VaoIdHouse);
	colorCode = 1;
	glUniform1i(colorCodeLocation, colorCode);
	myMatrix = matTranHouse;
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawElements(GL_TRIANGLES, 60, GL_UNSIGNED_BYTE, (void*)(0));

	// Drawing the shadow of the house
	colorCode = 2;
	glUniform1i(colorCodeLocation, colorCode);
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawElements(GL_TRIANGLES, 60, GL_UNSIGNED_BYTE, (void*)(0));

	// Drawing the house door
	glBindVertexArray(VaoIdDoor);
	colorCode = 1;
	glUniform1i(colorCodeLocation, colorCode);
	myMatrix = matTranDoor;
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawElements(GL_TRIANGLES, 132, GL_UNSIGNED_BYTE, (void*)(0));

	// Drawing the house windows
	glBindVertexArray(VaoIdWindow);
	colorCode = 1;
	glUniform1i(colorCodeLocation, colorCode);
	myMatrix = matTranWindowFront1;
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawElements(GL_TRIANGLES, 102, GL_UNSIGNED_BYTE, (void*)(0));
```

```
	myMatrix = matTranWindowFront2;
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawElements(GL_TRIANGLES, 102, GL_UNSIGNED_BYTE, (void*)(0));
	myMatrix = matTranWindowFront3;
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawElements(GL_TRIANGLES, 102, GL_UNSIGNED_BYTE, (void*)(0));
	myMatrix = matTranWindowFront4;
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawElements(GL_TRIANGLES, 102, GL_UNSIGNED_BYTE, (void*)(0));
	myMatrix = matTranWindowBack;
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawElements(GL_TRIANGLES, 102, GL_UNSIGNED_BYTE, (void*)(0));
	myMatrix = matTranWindowRightSide1 * matRotWindow;
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawElements(GL_TRIANGLES, 102, GL_UNSIGNED_BYTE, (void*)(0));
	myMatrix = matTranWindowRightSide2 * matRotWindow;
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawElements(GL_TRIANGLES, 102, GL_UNSIGNED_BYTE, (void*)(0));
	myMatrix = matTranWindowLeftSide1 * matRotWindow;
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawElements(GL_TRIANGLES, 102, GL_UNSIGNED_BYTE, (void*)(0));
	myMatrix = matTranWindowLeftSide2 * matRotWindow;
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawElements(GL_TRIANGLES, 102, GL_UNSIGNED_BYTE, (void*)(0));
	myMatrix = matTranWindowLeftSide3 * matRotWindow;
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawElements(GL_TRIANGLES, 102, GL_UNSIGNED_BYTE, (void*)(0));

	// Drawing the wind turbine
	glBindVertexArray(VaoIdTurbine);
	colorCode = 1;
	glUniform1i(colorCodeLocation, colorCode);
	myMatrix = matTranTurbine;
```

```
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glLineWidth(3.0);
	glDrawElements(GL_LINES, 8, GL_UNSIGNED_BYTE, (void*)(0));
	glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(8));

	// Drawing the shadow of the wind turbine
	colorCode = 2;
	glUniform1i(colorCodeLocation, colorCode);
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawElements(GL_LINES, 8, GL_UNSIGNED_BYTE, (void*)(0));
	glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(8));

	// Drawing the propeller of the wind turbine
	glBindVertexArray(VaoIdPropeller);
	colorCode = 1;
	glUniform1i(colorCodeLocation, colorCode);
	myMatrix = matTranPropeller * matRotPropeller;
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawElements(GL_LINES, 4, GL_UNSIGNED_BYTE, (void*)(0));
	glDrawElements(GL_TRIANGLES, 24, GL_UNSIGNED_BYTE, (void*)(4));

	// Drawing the shadow of the propeller of the wind turbine
	colorCode = 2;
	glUniform1i(colorCodeLocation, colorCode);
	glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
	glDrawElements(GL_LINES, 4, GL_UNSIGNED_BYTE, (void*)(0));
	glDrawElements(GL_TRIANGLES, 24, GL_UNSIGNED_BYTE, (void*)(4));

	glutSwapBuffers();
	glFlush();
}
void Cleanup(void)
{
```

```
	DestroyShaders();

	DestroyVBO();

}


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

{

	glutInit(&argc, argv);

	glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);

	glutInitWindowPosition(100, 100);

	glutInitWindowSize(1200, 900);

	glutCreateWindow("Desert scene");

	glewInit();

	Initialize();

	glutIdleFunc(RenderFunction);

	glutDisplayFunc(RenderFunction);

	glutKeyboardFunc(processNormalKeys);

	glutSpecialFunc(processSpecialKeys);

	glutCloseFunc(Cleanup);

	glutMainLoop();

}
```

Sasu Alexandru-Cristian
Grupa 342

Codul din shader-ul de fragment:

```
#version 330

in vec3 FragPos;
in vec3 Normal;
in vec3 inLightPos;
in vec3 inViewPos;
in vec3 dir;
in vec3 ex_Color;
in vec2 tex_Coord;

out vec4 out_Color;

uniform vec3 lightColor;
uniform int colorCode;
uniform sampler2D myTexture;
uniform sampler2D myNormalMap;

void main(void)
{
    if (colorCode == -1 || colorCode == 0 || colorCode == 1)  // For drawing textures, the
ground, objects
    {
          // Ambient
        float ambientStrength = 0.2f;
        vec3 ambient = ambientStrength * lightColor;

        // Diffuse
        vec3 normal;
        if (colorCode != -1 && colorCode != 0)
            normal = normalize(Normal);
        else
            normal = normalize(texture(myNormalMap, tex_Coord).xyz * 2.0f - 1.0f);
        vec3 lightDir = normalize(inLightPos - FragPos);
        // vec3 lightDir = normalize(dir);  // For directional light source
        float diff = max(dot(normal, lightDir), 0.0);
        vec3 diffuse = diff * lightColor;

        // Specular
        float specularStrength = 0.5f;
        vec3 viewDir = normalize(inViewPos - FragPos);  // Vector towards normalized observer (V)
        vec3 reflectDir = reflect(-lightDir, normal);  // Reflection of light ray (R)
        float spec = pow(max(dot(viewDir, reflectDir), 0.0), 1);
        vec3 specular = specularStrength * spec * lightColor;

        vec3 emission = vec3(0.0, 0.0, 0.0);

        if (colorCode == 0 || colorCode == 1)  // For the ground and objects
        {
            vec3 result = emission + (ambient + diffuse + specular) * ex_Color;
            out_Color = vec4(result, 1.0f);
        }
        else
            if (colorCode == -1)  // For textures
            {
                vec4 result = vec4(emission + (ambient + diffuse + specular), 1.0f) *
texture(myTexture, tex_Coord);
                out_Color = result;
```

```
            }
    }
    else
        if (colorCode == 2)  // For drawing the shadow of objects
        {
            vec3 black = vec3 (0.0, 0.0, 0.0);
              out_Color = vec4(black, 1.0);
        }
}
```

Sasu Alexandru-Cristian
Grupa 342

Codul din shader-ul de vârf:

```
#version 330

layout(location=0) in vec4 in_Position;
layout(location=1) in vec3 in_Color;
layout(location=2) in vec3 in_Normal;
layout(location=3) in vec2 texCoord;

out vec3 FragPos;
out vec3 Normal;
out vec3 inLightPos;
out vec3 inViewPos;
out vec3 ex_Color;
out vec3 dir;
out vec2 tex_Coord;

uniform mat4 matShadow;
uniform mat4 myMatrix;
uniform mat4 view;
uniform mat4 projection;
uniform vec3 lightPos;
uniform vec3 viewPos;
uniform vec3 lightColor;
uniform int colorCode;

void main(void)
{
    ex_Color = in_Color;
    tex_Coord = vec2(texCoord.x, 1 - texCoord.y);

       if (colorCode == -1 || colorCode == 0 || colorCode == 1)  // For drawing textures, the
ground, objects
    {
             gl_Position = projection * view * myMatrix * in_Position;
        Normal = mat3(projection * view * myMatrix) * in_Normal;
        inLightPos = vec3(projection * view * myMatrix * vec4(lightPos, 1.0f));
        inViewPos = vec3(projection * view * myMatrix * vec4(viewPos, 1.0f));
        dir = mat3(projection * view * myMatrix) * vec3(0.0,100.0,200.0);  // For directional
light source
    }
    else
          if (colorCode == 2)  // For drawing the shadow of objects
                gl_Position = projection * view * matShadow * myMatrix * in_Position;

    FragPos = vec3(gl_Position);
}
```

Sasu Alexandru-Cristian
Grupa 342

Fișierul stb_image.h:

Conținutul său poate fi obținut de la link-ul:
https://raw.githubusercontent.com/nothings/stb/master/stb_image.h

Imagini:

Imaginile utilizate pot fi obținute de la următorul link de Google Drive:
https://drive.google.com/drive/folders/1pO2rEM_7GdyGoJCuY8ulY8y2sHwDnCva?usp=share_link