EEL5771 – Engineering Applications of Computer Graphics
Project Description

In this project, the input to your program is an environment file that may include:

- Name of a height map file (a pgm file to generate height values for a regular mesh)
- a texture file name (a bmp file to cover the given terrain)
- multiple object definitions
- multiple billboard object definitions
- a skydome definition
- a water level definition
- a cloud plane definition
- a flare sun definition.

The format of the file will be as follows (Bold words are keywords; italic words are parameters of mentioned type); Each line must start with a keyword (**HEIGHT, TEXTURE, LENGTHS, OBJ, BILLBOARD, SKYDOME, WATER, CLOUDS, SUN**) followed by parameters for each type.
Only height map is mandatory. If there is no texture file you are required to assign a color value to each vertex of the mesh according to its height value (Maximum height will be blue, and vertices at minimum height will be colored in black).

Parameters and Format of the input environment File:
Basic environment will be as follows:

HEIGHT *HeightMapFile MIN_HEIGHT MAX_HEIGHT*
TEXTURE *TextureFile*
LENGTHS *XLEN YLEN*

OBJ *OBJ_TYPE OBJFILE XPOS YPOS*
…
BILLBOARD *BILLBOARDNAME BILLBOARD_BMP_FILE XPOS1 YPOS1 XPOS2 YPOS2*
….

SKYDOME *SkyTexture ROTATIONANGLE_PER_REFRESH*

WATER *WaterTexture Z_WATER_PLANE Z_OSSILATION_WATER_PLANE*

SUN *SunTexture1 SunTexture2*

Now let's look at the different blocks in this file format:

1. **Terrain definitions :**
   HEIGHT *HeightMapFile MIN_HEIGHT MAX_HEIGHT*
   TEXTURE *TextureFile*
   LENGTHS *XLEN YLEN*

HeightMapFile: name of a pgm file. pgm file will include some integer values .

Texturefile : name of the texture bitmap that covers the terrain.
XLEN YLEN: lengths of terrain cells in X and Y directions.

You are required to generate a height map according to the MIN_HEIGHT and MAX_HEIGHT values given in the definition file. That is

"**HEIGHT** *mymap.pgm –10 200*"
TEXTURE *mytex.bmp*
5 4

means that you will read mymap.pgm. You will assign height value of –10 to the edges with smallest color, and 200 to the edges with maximum color value. The other height assignments will change linearly according to the color value. And you will cover each quad of the mesh with the given texture file (You are free to cover all terrain with one bmp or apply the given texture to each quad separately.) The width of each quad of the mesh will be 5 units along x and 4 units along y.

2. Object and Billboards Definitions :

OBJ *OBJ_TYPE OBJFILE XPOS YPOS*

OBJ_TYPE OBJFILE XPOS YPOS

OBJ_TYPE may be **Static** or **Animated**. If it is a **Static**, *OBJFILE* will be the name of the *object file*(such as *space_ship.mod*). If the type is *Animated*, then it will be a *framed* one with *filename+[frame_number].mod* will be the model file for frame numbered with the attached digits. Format of the mod files is explained below. *XPOS* and *YPOS* will be column and row locations for given object [a cell will include one object or may overlap others]. That is;

OBJ *Static tree.mod 10 5*

Means that you will load a mod file called *tree.mod* and object will be rendered at column 10, row 5. You are required to place the object on to the terrain.

OBJ Animated XYZ.*mod 10 5*

Means that you will load a mod file called *XYZ.mod* and object will be rendered at column 10, row 5. You are required to place the object on to the terrain. X will start from 01 and you will change the model file loaded by incrementing the counter by 1 at each step (to increase the speed you have to load all possible files for the given object in preprocessing step and generate call lists for each frame.)

BILLBOARD *BILLBOARDNAME BILLBOARD_BMP_FILE XPOS1 YPOS1 XPOS2 YPOS2 ….*

Billboards will be composed of 2 crossed rectangles facing the camera, each having texture of given bmp file (*BILLBOARD_BMP_FILE*) with ALPHA Blending enabled. There may be multiple X and Y values after the definition that means that you will put that billboard to those locations during rendering.

3. Other definitions:

SKYDOME *SkyTexture ROTATIONANGLE_PER_REFRESH*

You will render a *gluSphere* covering the given terrain, sphere will have the texture of the mentioned bmp file. And during each rendering step you will rotate the sphere around z axis by the mentioned angle.

WATER *WaterTexture HEIGHT_WATER_PLANE Z_OSSILATION_WATER_PLANE*

For simple water effect, you will make use of mesh rendering trick. You will render a regular mesh (as you do for the terrain) with each vertex having height of *HEIGHT_WATER_PLANE* initially. While rendering you will increment and decrement this height value down to *HEIGHT_WATER_PLANE -Z_OSSILATION_WATER_PLANE* units and up to *HEIGHT_WATER_PLANE +Z_OSSILATION_WATER_PLANE* units. .You will cover each quad of this mesh by the given water texture. Do not forget using transparency (Texture will have alpha values again like billboard textures)

SUN *SunTexture1 SunTexture2*

It is simple just a rectangle with alpha blending enabled textures, you will randomize current texture from the given bmp texture list during rendering. Place the sun to a place of your choose (make it visible at least ☺).

4. Controls:

- Initially you will place the camera in your system such that you will be looking from (1,1) towards (N,M) cells of the terrain. Mouse will control movement, moving the mouse forward will move the camera forward and moving it backward will move the camera backward. By moving mouse while left mouse button is clicked you will be able to change the direction of the camera in XY plane (head), and by moving the camera while right mouse button is clicked you will be able to change the direction of the camera in z dimension (pitch). There won't be any roll operations

- By pressing 'S' button, the user will be able to scale all objects (objects, billboards) by 2

- By pressing 's' button, the user will be able to scale all objects (objects, billboards) by ½

- The user may wish to go directly to another cell. In that case she will press 'G' or 'g', and program will prompt her to input a 2-digit number for column and row numbers of the transfer.

- 'L' or 'l' will disable/enable lighting. Place the light in a good position☺.

- 'F' or 'f' will disable/enable fog. You are free to choose fog parameters.

Format of pgm file

pgm(5) pgm(5)

NAME

pgm - portable graymap file format

DESCRIPTION

The portable graymap format is a lowest common denominator grayscale file format. The definition is as follows:

- A "magic number" for identifying the file type. A pgm file's magic number is the two characters "P2".

- Whitespace (blanks, TABs, CRs, LFs).

- A width, formatted as ASCII characters in decimal.

- Whitespace.

- A height, again in ASCII decimal.

- Whitespace.

- The maximum gray value, again in ASCII decimal.

- Whitespace.

- Width * height gray values, each in ASCII decimal, between 0 and the specified maximum value, separated by whitespace, starting at the top-left corner of the graymap, proceeding in normal English reading order. A value of 0 means black, and the maximum value means white.

- Characters from a "#" to the next end-of-line are ignored (comments).

- No line should be longer than 70 characters.

Here is an example of a small graymap in this format:

```
P2
# feep.pgm
24 7
15
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
 0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
 0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
 0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
 0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Programs that read this format should be as lenient as possible, accepting anything that looks remotely like a graymap. Below is an example.

```
int maxval;
typedef struct
{
        double **real;
        int width;
        int height;
} image_type;

image_type *read_image(image_type *image, char *filename)
{
        FILE *inf;
        char temp[256];
        int width, height,color;

        if((inf = fopen(filename, "r+")) == NULL)
        {
                printf("Unable to open pgm file: %s!\n", filename);
                exit(1);
        }
        if(fscanf(inf, " P2 "))
        {
                printf("not a valid pgm: %s!\n",filename);
                exit(1);
        }

        fgets(temp, 128 * sizeof(char), inf);
        fscanf(inf, " %d %d %d", &width, &height, &maxval);
        image = new_image(image, width, height, maxval);

        for(i=0; i<image->height; i++)
        {
                for(j=0; j<image->width; j++)
                {
                        fscanf(inf, " %d", &color);
                        image->real[i][j] = (double) color;
                }
        }

        fclose(inf);
        return image;
}
```

Format of the mod file

This format is simple and is generated by dumping data of models such that. # Normal Texture and Vertex group will be a triangle. Your job is to render them in a GL_TRIANGLES block. Detailed format of 3-line blocks are :

"N %.3f %.3f %.3f\t" for normal
"T %.3f %.3\t" for texture
"V %.3f %.3f %.3f\n" for vertex

Which corresponds to

glNormal3f (normal[0],normal[1],normal[2]);
glTexCoord2f(textures[0],textures[1]);
glVertex3f (vertex[0],vertex[1],vertex[2]);
(code group for one vertex)

Here is a Sample input File

HEIGHT *map.pgm 0 10*
LENGTHS *3 3*
OBJ Animated anim.bal 1 1
BILLBOARD *CamAgaci agac.bmp 2 1 2 2 3 3 5 5 100 100*
SKYDOME *Sky0.bmp 3*
WATER *Water1.bmp 3 0.1*
SUN *nebula.bmp star.bmp*

I have also included some extra code sample, and some textures and model dump files.