5. Импортирование созданных моделей и ландшафта в программную среду

Экспортировать файл модели будем в формате obj. Данный формат содержит информацию о вершинах модели и гранях.

Формат obj файла:

#вершины

v 0.123 0.234 0.345

v -0.123 0.234 0.345

v 0.123 -0.234 0.345

#грани (индексы вершин)

f 1 2 3

f 3 4 5

f 6 3 7

f 7 8 9

Алгоритм чтения из файла:

Токен - это строка с присвоенным и, таким образом, идентифицированным значением. В нашем случае токенами будут “v”, “t”, “#”, а также некоторые другие лексемы obj формата, не используемые нашей программой, но которые могут быть сгенерированы программой 3D моделирования. Их мы будем игнорировать.

Объявление токена:

enum TokenKind {  
 *TOKEN\_FLOAT* = 128,*//0-128 reserved for symbols  
 TOKEN\_INT*,  
 *TOKEN\_NAME*,  
 *TOKEN\_COMMENT*,  
 *TOKEN\_VERTEX*,  
 *TOKEN\_FACE*,  
 *TOKEN\_OBJECT\_NAME*,  
 *// ...*};

struct Token {  
 TokenKind kind;  
 const char \*start;  
 const char \*end;  
 union {  
 double float\_val;  
 uint32\_t int\_val;  
 const char\* name;  
 };  
};

Центральная функция – next\_token(), с помощью которой, итеративно проходя по всему файлу символ за символом, мы получаем информацию о следующем токене. Так же в ней, считываются числовые значения координат и индексов вершин.

Загрузка файла в память:

const char\* stream\_start;  
const char\* stream;

void init\_stream(const char\* str) {  
 stream\_start = stream = str;  
 char\_to\_digit['0'] = 0;  
 char\_to\_digit['1'] = 1;  
 char\_to\_digit['2'] = 2;  
 char\_to\_digit['3'] = 3;  
 char\_to\_digit['4'] = 4;  
 char\_to\_digit['5'] = 5;  
 char\_to\_digit['6'] = 6;  
 char\_to\_digit['7'] = 7;  
 char\_to\_digit['8'] = 8;  
 char\_to\_digit['9'] = 9;  
  
 next\_token();  
}  
  
void init\_stream\_from\_file(FILE\* from){  
 size\_t grow\_size = 1024;  
 char\* buffer = (char\*)malloc(grow\_size);  
 int len = 0,grow\_len = 0;  
  
 int end = 0;  
 for(;end!=**EOF**;){  
 end = fscanf(from,"%c",buffer+len);  
 len++;  
 grow\_len++;  
 if (grow\_len == grow\_size){  
 buffer = (char\*)realloc(buffer,len+grow\_size);  
 grow\_len = 1;  
 }  
 }  
 buffer[len-1] = '\0';  
 init\_stream(buffer);  
}

Функция parse\_vertex, будет вызываться каждый раз, когда находится токен вершины. Функция последовательно считывает координаты вершин и записывает из в вектор.

POINT3 vertex;  
Edge edge;  
std::vector<uint32\_t> indexes;  
  
bool parse\_vertex(){  
 next\_token();  
 expect\_token(TokenKind(' '));  
 if(is\_token(*TOKEN\_FLOAT*)){  
 vertex.x = token.float\_val;  
 }else if(is\_token(*TOKEN\_INT*)){  
 vertex.x = token.int\_val;  
 }else{  
 fatal("unexpected token");  
 }  
  
 next\_token();  
 expect\_token(TokenKind(' '));  
 if(is\_token(*TOKEN\_FLOAT*)){  
 vertex.y = token.float\_val;  
 }else if(is\_token(*TOKEN\_INT*)){  
 vertex.y = token.int\_val;  
 }else{  
 fatal("unexpected token");  
 }  
  
 next\_token();  
 expect\_token(TokenKind(' '));  
 if(is\_token(*TOKEN\_FLOAT*)){  
 vertex.z = token.float\_val;  
 }else if(is\_token(*TOKEN\_INT*)){  
 vertex.z = token.int\_val;  
 }else{  
 fatal("unexpected token");  
 }  
  
 next\_token();  
 expect\_token(TokenKind('\n'));  
 if (is\_token(*TOKEN\_VERTEX*)){  
 return false;  
 }else{  
 return true;  
 }  
}

Функция parse\_edge, будет вызываться каждый раз, когда находится токен грани. Функция последовательно считывает индексы вершин и записывает новую грань в вектор.

bool parse\_edge(){  
 next\_token();  
 int count = 0;  
 while(token.kind != '\n' && \*(stream+1) !='\0'){  
 expect\_token(TokenKind(' '));  
 if (is\_token(*TOKEN\_INT*)){  
 indexes.push\_back(token.int\_val-1);  
 count++;  
 }else{  
 fatal("unexpected token");  
 }  
 next\_token();  
 if(is\_token(TokenKind('/'))){  
 while(token.kind != TokenKind(' ') && token.kind != TokenKind('\n') && \*(stream+1) !='\0'){  
 next\_token();  
 }  
 }  
 }  
 edge.vertices\_count = count;  
  
 next\_token();  
 if (is\_token(*TOKEN\_FACE*)){  
 return false;  
 }else{  
 return true;  
 }  
}