Библиотека *Support Raytracing Library*

Библиотека *Support Raytracing Library* построена на базе библиотек *Base Math Library* и *Base Render Library* и содержит необходимые классы и функции для визуализации произвольных компьютерных сцен методом трассировки лучей на графическом процессоре. Данная библиотека включает в себя классы для описания источников света и материалов на основе модели освещения Уиттеда, классы базовых геометрических объектов (сфера, прямоугольник, куб) и загружаемых из файлов в формате OBJ трехмерных моделей, а также классы для формирования ускоряющей структуры и передачи данных на графический процессор.

1. Класс *Light*

Таблица 1. Поля и методы класса *Light*

Открытые поля	
Синтаксис	*Описание*
`Vector3D Ambient`	*Интенсивность фонового света*

`Vector3D Diffuse`	*Интенсивность диффузного света*
`Vector3D Specular`	*Интенсивность зеркального света*
`Vector3D Position`	*Положение источника света*
`unsigned Number`	*Номер источника света*
Конструктор	
Синтаксис	*Описание*
`Light ( unsigned number = 0, const Vector3D& position = Vector3D ( 10.0F, 10.0F, 10.0F ), const Vector3D& ambient = Vector3D ( 0.1F, 0.1F, 0.1F ), const Vector3D& diffuse = Vector3D ( 0.6F, 0.6F, 0.6F ), const Vector3D& specular = Vector3D ( 0.6F, 0.6F, 0.6F ) )`	*Создает новый точечный источник света номер number с заданным положением position, а также интенсивностями фонового, диффузного и зеркального света ambient, diffuse и specular соответственно*
Вспомогательные функции	
Синтаксис	*Описание*
`void Setup ( void )`	*Передает OpenGL параметры точечного источника света*
`void SetShaderData ( ShaderManager * manager )`	*Передает необходимые данные активной шейдерной программе*
Функции рисования	
Синтаксис	*Описание*
`void Draw ( void )`	*Выполняет рисование точечного источника света средствами OpenGL*

2. Класс `Material`

Таблица 2. Поля и методы класса `Material`

Открытые поля	
Синтаксис	*Описание*
`Vector3D Ambient`	*Коэффициент отражения фонового света*
`Vector3D Diffuse`	*Коэффициент отражения диффузного света*
`Vector3D Specular`	*Коэффициент отражения зеркального света*
`float Shininess`	*Коэффициент резкости бликов*
`Vector3D Reflection`	*Вклад отраженных лучей*
`Vector3D Refraction`	*Вклад преломленных лучей*
`float Density`	*Коэффициент преломления материала*

`float Dissolve`	*Коэффициент прозрачности (при расчете прозрачности лучи не преломляются)*
`Texture2D * Texture`	*Текстура материала*
`Vector2D Scale`	*Коэффициенты масштабирования текстуры*
`int Identifier`	*Идентификатор материала*
Конструктор	
Синтаксис	*Описание*
`Material (const Vector3D& ambiant = Vector3D (0.2F, 0.2F, 0.2F), const Vector3D& diffuse = Vector3D (0.8F, 0.8F, 0.8F), const Vector3D& specular = Vector3D (0.8F, 0.8F, 0.8F), float shininess = 32.0F, const Vector3D& reflection = Vector3D :: Zero, const Vector3D& refraction = Vector3D :: Zero, float density = 1.5F, float dissolve = 1.0F, Texture2D * texture = NULL, const Vector2D& scale = Vector2D :: Unit)`	*Создает новый материал с заданными коэффициентами модели освещения Фонга (ambient, diffuse, specular и shininess), отражающими и преломляющими свойствами reflection и refraction, коэффициентом преломления density, прозрачностью dissolve, текстурой материала texture и коэффициентами масштабирования текстуры scale*
Вспомогательные функции	
Синтаксис	*Описание*
`void Setup ( void )`	*Передает OpenGL параметры материала (в рамках модели локального освещения Фонга)*

3. Класс `Vertex`

Таблица 3. Поля и методы класса `Vertex`

Открытые поля	
Синтаксис	*Описание*
`Vector3D Position`	*Положение вершины*
`Vector3D Normal`	*Нормаль в вершине*
`Vector2D TexCoord`	*Текстурные координаты вершины*
Конструктор	
Синтаксис	*Описание*
`Vertex ( const Vector3D& position = Vector3D :: Zero, const Vector3D& normal = Vector3D :: AxisZ, const Vector2D& texcoord = Vector2D :: Zero )`	*Создает новую вершину с заданным положением position, нормалью normal и текстурными координатами texcoord*
Функции рисования	
Синтаксис	*Описание*

`void Draw ( void )`	*Выполняет рисование вершины средствами OpenGL*

4. Класс *Triangle*

Таблица 4. Поля и методы класса ***Triangle***

Открытые поля	
Синтаксис	*Описание*
`Vertex * VertexA`	*Первая вершина (A) треугольника*
`Vertex * VertexB`	*Вторая вершина (B) треугольника*
`Vertex * VertexC`	*Третья вершина (C) треугольника*
`Material * Properties`	*Свойства материала объекта, к которому принадлежит данный треугольник*
Конструктор	
Синтаксис	*Описание*

`Triangle ( Vertex * vertexA,`
`Vertex * vertexB,`
`Vertex * vertexC,`
`Material * properties )`
	Создает новый треугольник с вершинами vertexA, vertexB и vertexC и заданными свойствами материала properties

Функции рисования	
Синтаксис	*Описание*
`void Draw ( void )`	*Выполняет рисование треугольника средствами OpenGL*
Дополнительные функции	
Синтаксис	*Описание*
`bool IsEmpty ( void )`	*Возвращает **true**, если площадь треугольника близка к нулю (и он может быть отброшен перед визуализацией)*
`Vector3D GetNormal ( void )`	*Возвращает нормаль к плоскости треугольника* $n = (AB \times AC)$
`Vector3D GetMinimum ( void )`	*Возвращает минимальную точку ограничивающего параллелепипеда треугольника*
`Vector3D GetMaximum ( void )`	*Возвращает максимальную точку ограничивающего параллелепипеда треугольника*

5. Класс `Primitive`

В библиотеке *Support Raytracing Library* определяется несколько классов объектов для визуализации, которые связаны между собой отношением наследования.

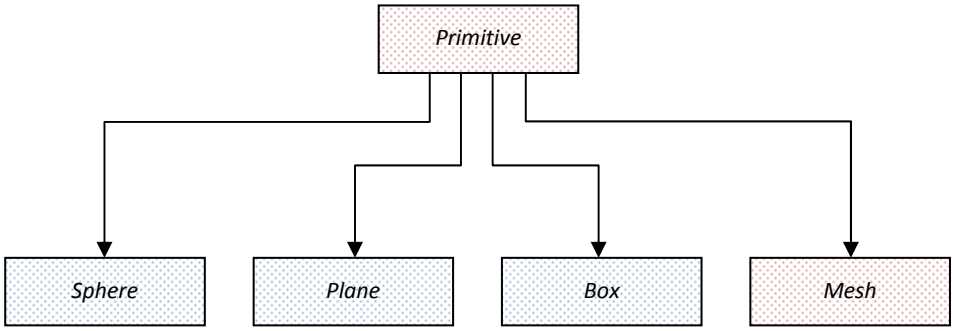

Рис. 1. Иерархия объектов для визуализации

Базовым классом является *абстрактный* класс `Primitive`, который определяет общие для всех объектов поля и методы: применяемое аффинное преобразование, свойства материала, список треугольников, функции тесселяции и рисования и некоторые другие. От базового класса наследуются такие объекты, как сфера (класс `Sphere`), прямоугольник (класс `Plane`), параллелепипед (класс `Box`), а также произвольная загружаемая из файла модель (класс `Mesh`).

Таблица 5. Поля и методы класса `Primitive`

Открытые поля	
Синтаксис	*Описание*
`vector < Triangle * > Triangles`	*Список треугольников объекта*
`Transform * Transformation`	*Аффинное преобразование объекта*
`Material * Properties`	*Свойства материала объекта*
`const char * Name`	*Имя объекта*
`bool Visible`	*Флаг видимости*
Конструктор	
Синтаксис	*Описание*
`Primitive ( Transform * transformation = new Transform ( ), Material * properties = new Material ( ), const char * name = "Primitive", bool visible = true )`	*Создает новый абстрактный объект с заданным аффинным преобразованием transformation, свойствами материала properties, именем name и флагом видимости visibility*
Функции тесселяции	

Синтаксис	Описание
`virtual void Tesselate ( void ) = 0`	Выполняет тесселяцию объекта (генерирует треугольники)
Функции рисования	
Синтаксис	**Описание**
`void Draw ( void )`	Выполняет рисование объекта средствами OpenGL

6. Класс *Sphere*

Таблица 6. Поля и методы класса ***Sphere*** (отличные от класса ***Primitive*** поля и методы)

Открытые поля	
Синтаксис	**Описание**
`float Radius`	Радиус сферы
`int Slices`	Число разбиений по широте
`int Stacks`	Число разбиений по долготе
Конструктор	
Синтаксис	**Описание**
`Sphere ( float radius = 1.0F, int slices = 25, int stacks = 25, Transform * transformation = new Transform ( ), Material * properties = new Material ( ), const char * name = "Sphere", bool visible = true )`	Создает новую сферу с радиусом radius, числом разбиений по широте и долготе slices и stacks соответственно, аффинным преобразованием transformation, свойствами материала properties, именем name и флагом видимости visibility

7. Класс *Plane*

Таблица 7. Поля и методы класса ***Plane*** (отличные от класса ***Primitive*** поля и методы)

Открытые поля	
Синтаксис	**Описание**
`Vector2D Radius`	Половинный размер прямоугольника
Конструктор	
Синтаксис	**Описание**
`Plane ( const Vector2D& radius = Vector2D :: Unit, Transform * transformation = new Transform ( ), Material * properties = new Material ( ), const char * name = "Plane", bool visible = true )`	Создает новый прямоугольник с половинным размером radius, аффинным преобразованием transformation, свойствами материала properties, именем name и флагом

	видимости visibility

8. Класс *Box*

Таблица 8. Поля и методы класса ***Box*** (отличные от класса ***Primitive*** поля и методы)

Открытые поля	
Синтаксис	*Описание*
`Vector3D Radius`	*Половинный размер параллелепипеда*
Конструктор	
Синтаксис	*Описание*

`Box ( const Vector3D& radius = Vector3D :: Unit,`
`Transform * transformation = new Transform ( ),`
`Material * properties = new Material ( ),`
`const char * name = "Plane",`
`bool visible = true )`
	Создает новый параллелепипед с половинным размером radius, аффинным преобразованием transformation, свойствами материала properties, именем name и флагом видимости visibility

9. Класс *Mesh*

Таблица 9. Поля и методы класса ***Mesh*** (отличные от класса ***Primitive*** поля и методы)

Открытые поля	
Синтаксис	*Описание*
`const OBJModel * Model`	*Трехмерная модель, загружаемая из файла в формате OBJ. Может состоять из нескольких групп треугольников с различными материалами*
`int Group`	*Номер группы треугольников для визуализации*
Конструктор	
Синтаксис	*Описание*

`Mesh ( const OBJModel * model,`
`int group = 0,`
`Transform * transformation = new Transform ( ),`
`Material * properties = new Material ( ),`
`const char * name = "Plane",`
`bool visible = true )`
	Создает новую сетку треугольников для заданной группы group трехмерной модели model, с заданным аффинным преобразованием transformation, свойствами материала properties, именем name и флагом видимости visibility. Если в качестве свойств материала properties передано значение NULL, то используется соответствующий материал трехмерной модели model

10. Классы *Voxel* и *UniformGrid*

Таблица 10. Поля и методы класса ***Voxel***

Открытые поля	
Синтаксис	*Описание*
`Vector3D Position`	*Положение центральной точки вокселя*
`Vector3D Radius`	*Половинный размер вокселя*
`vector < Triangle * > Triangles`	*Список треугольников, перекрывающихся с данным вокселем*
Конструктор	
Синтаксис	*Описание*
`Voxel ( const Vector3D& position, const Vector3D& radius )`	*Создает новый воксель с заданным положением position и половинным размером radius*

Таблица 11. Поля и методы класса ***UniformGrid***

Конструктор	
Синтаксис	*Описание*

`UniformGrid ( int partitionsX = 16,`
`int partitionsY = 16,`
`int partitionsZ = 16 )`
	Создает новую равномерную сетку с заданным числом разбиений

Функции формирования и доступа к равномерной сетке	
Синтаксис	*Описание*
`void BuildGrid ( Volume * volume, vector < Triangle * > triangles )`	*Генерирует равномерную сетку для заданного ограничивающего объема volume и заданного списка треугольников triangles*
`Voxel * GetVoxel ( int x, int y, int z )`	*Возвращает воксель равномерной сетки с номером (x, y, z)*
Функции чтения параметров равномерной сетки	
Синтаксис	*Описание*
`int GetPartitionsX ( void )`	*Возвращает число разбиений по оси x*
`int GetPartitionsY ( void )`	*Возвращает число разбиений по оси y*
`int GetPartitionsZ ( void )`	*Возвращает число разбиений по оси z*

11. Классы *Volume* и *Scene*

Таблица 12. Поля и методы класса ***Volume***

Открытые поля	
Синтаксис	*Описание*
`Vector3D Minimum`	*Минимальная точка ограничивающего параллелепипеда сцены*
`Vector3D Maximum`	*Максимальная точка ограничивающего параллелепипеда сцены*
`Vector3D Color`	*Цвет линий для рисования ограничивающего параллелепипеда сцены*
`bool Visible`	*Флаг видимости*
Конструктор	
Синтаксис	*Описание*
`Volume ( const Vector3D& minimum = Vector3D ( -5.0F, -5.0F, -5.0F ), const Vector3D& maximum = Vector3D ( 5.0F, 5.0F, 5.0F ), const Vector3D& color = Vector3D ( 0.6F, 0.6F, 0.6F ), bool visible = true )`	*Создает новый ограничивающий параллелепипед сцены с заданной минимальной и максимальной точкой minimum и maximum соответственно, цветом color и флагом видимости visible*
Функции рисования	
Синтаксис	*Описание*
`void Draw ( void )`	*Выполняет рисование ограничивающего параллелепипеда сцены средствами OpenGL*

Таблица 13. Поля и методы класса ***Scene***

Открытые поля	
Синтаксис	*Описание*
`vector < Light * > Lights`	*Список источников света (не более 8)*
`vector < Primitive * > Primitives`	*Список объектов*
`Camera * Viewer`	*Камера для 'съемки' сцены*
`Volume * Box`	*Ограничивающий параллелепипед сцены*
`UniformGrid * Grid`	*Равномерная сетка для ускорения поиска пересечений*
Конструктор	
Синтаксис	*Описание*
`Scene ( Camera * viewer = new Camera ( ), Volume * volume = new Volume ( ) )`	*Создает новую сцену с пустыми списками объектов и источников света, заданной камерой viewer и ограничивающим параллелепипедом volume*

Вспомогательные функции	
Синтаксис	*Описание*
`void BuildGrid ( int partitionsX, int partitionsY, int partitionsZ )`	*Генерирует равномерную сетку с заданным числом разбиений для данной сцены*
`void SetShaderData ( ShaderManager * manager )`	*Передает необходимые данные активной шейдерной программе*
Функции рисования	
Синтаксис	*Описание*
`void Draw ( void )`	*Выполняет рисование сцены средствами OpenGL*

12. Класс `StaticData`

Таблица 14. Поля и методы класса `StaticData`

Открытые поля	
Синтаксис	*Описание*
`Texture3D * VoxelTexture`	*Трехмерная текстура для хранения равномерной сетки*
`Texture2D * PositionTexture`	*Двумерная текстура для хранения положений вершин*
`Texture2D * NormalTexture`	*Двумерная текстура для хранения нормалей в вершинах*
`Texture2D * TexCoordTexture`	*Двумерная текстура для хранения текстурных координат*
`Texture1D * MaterialTexture`	*Одномерная текстура для хранения материалов*
Конструктор	
Синтаксис	*Описание*
`StaticData ( void )`	*Создает новый класс для передачи статической информации о сцене на графический процессор*
Функции формирования статической информации	
Синтаксис	*Описание*
`void SetupTextures ( Scene * scene )`	*Формирует статическую информацию (равномерная сетка, атрибуты вершин, материалы) для заданной сцены*
`void SetShaderData ( ShaderManager * manager )`	*Передает статическую информацию активной шейдерной программе*