Библиотека *Base Render Library*

Библиотека *Base Render Library* содержит базовые классы для разработки произвольных приложений трехмерной графики на базе интерфейса OpenGL. Библиотека включает в себя объектные “обертки” для работы с вершинными и фрагментными *шейдерными программами*, *текстурными объектами*, а также *буфером кадра* (используется для вывода результата визуализации в текстуру). Кроме того, в данной библиотеке определяется класс *камеры*, с помощью которого устанавливается положение и ориентация наблюдателя на компьютерной сцене. Далее приводится описание принятой модели камеры.

1. Модель камеры

Фактически камера представляет собой систему координат, то есть обладает своим *положением* (*Position*), направлением *взгляда* (*View*), направлениями *вверх* (*Up*) и *вбок* (*Side*), в качестве которого выбирается направление *влево* или *вправо*.

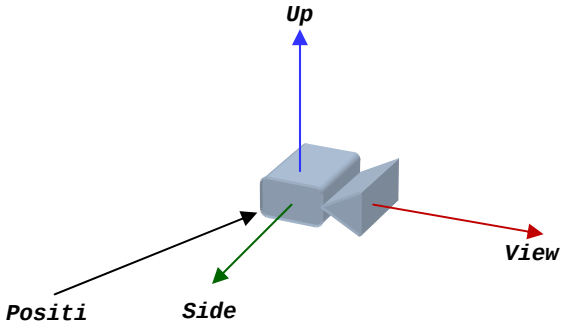

Рис. 1. Координатная система камеры

При этом направления взгляда, вправо и вверх образуют *ортонормированный базис*, то есть они попарно перпендикулярны и их длины равны единице.

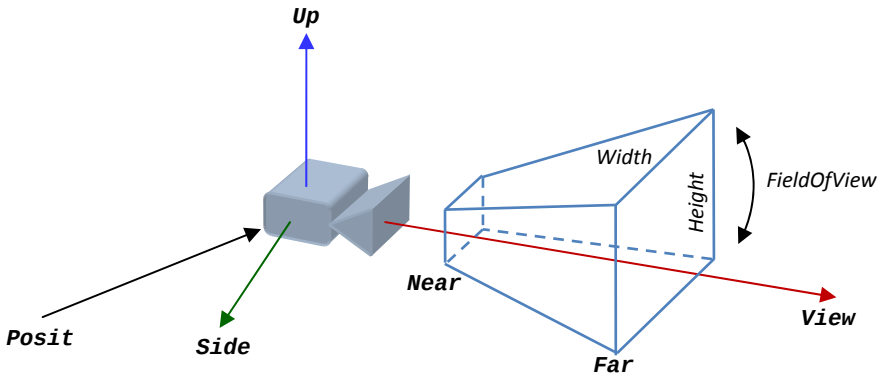

Рис. 2. Усеченная пирамида видимости для камеры

Поскольку камера используется для визуализации, это не просто координатная система. С ней также связана *усеченная пирамида видимости* (*view frustum*). При этом вектор взгляда задает направление этой пирамиды и ее основания, а векторы вправо и вбок – ее боковые стороны.

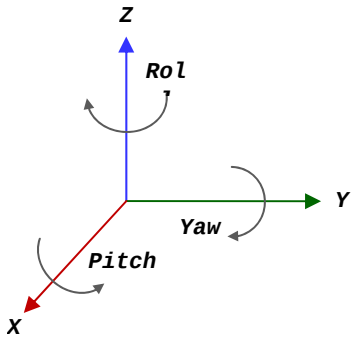

Рис. 3. Задние ориентации системы координат углами Эйлера

Пирамида видимости характеризуется *углом раствора* (`FieldOfView`), а также *ближним* и *дальним* расстояниями отсечения (`Near` и `Far`). Кроме того, удобно добавить к свойствам камеры *размеры окна* (`Width` и `Height`), в которое осуществляется рендеринг.

Следует также отметить, что задавать ориентацию камеры при помощи трех ортонормированных векторов на практике не очень удобно. Гораздо удобнее использовать в этих целях *углы Эйлера* – именно такой способ был принят в рассматриваемой реализации.

2. Класс `Camera`

Таблица 1. Поля и методы класса `Camera`

Конструктор	
Синтаксис	*Описание*
`Camera ( const Vector3D& pos = Vector3D ( 0.0F, 0.0F, -10.0F ), const Vector3D& orient = Vector3D :: Zero )`	*Создает новую камеру с заданным положением pos и углами поворота относительно осей системы координат orient*
Функции изменения положения камеры	
Синтаксис	*Описание*
`void MoveLocal ( float delta, const Vector3D& dir )`	*Перемещает камеру на расстояние delta вдоль направления dir в **локальной** системе координат*
`void MoveWorld ( float delta, const Vector3D& dir )`	*Перемещает камеру на расстояние delta вдоль направления dir в **мировой** системе координат*
Функции изменения ориентации камеры	
Синтаксис	*Описание*
`void RotateLocal ( float angle, const Vector3D& dir )`	*Поворачивает камеру на угол angle вокруг направления dir в **локальной** системе координат*
`void RotateWorld ( float angle, const Vector3D& dir )`	*Поворачивает камеру на угол angle вокруг направления dir в **мировой** системе координат*
Функции установки окна вывода и усеченной пирамиды видимости	
Синтаксис	*Описание*
`void SetViewport ( unsigned int width = 512, unsigned int height = 512 )`	*Устанавливает новую ширину width и высоту height окна вывода*
`void SetFrustum ( float fov = ONEPI / 3.0F, float near = 0.01F, float far = 1000000.0F )`	*Устанавливает новые параметры усеченной пирамиды видимости: угол раствора камеры fov, расстояния до ближней near и дальней far плоскости отсечения*
Функции чтения параметров камеры	
Синтаксис	*Описание*

`unsigned int GetWidth ( void )`	*Возвращает ширину окна вывода*
`unsigned int GetHeight ( void )`	*Возвращает высоту окна вывода*
`float GetFieldOfView ( void )`	*Возвращает угол раствора камеры*
`float GetNearPlane ( void )`	*Возвращает расстояние до ближней плоскости отсечения*
`float GetFarPlane ( void )`	*Возвращает расстояние до дальней плоскости отсечения*
`const Vector3D& GetPosition ( void )`	*Возвращает положение камеры*
`const Vector3D& GetSide ( void )`	*Возвращает вектор направления вбок*
`const Vector3D& GetUp ( void )`	*Возвращает вектор направления вверх*
`const Vector3D& GetView ( void )`	*Возвращает вектор направления взгляда*
`const Matrix3D& GetWorldToCamera ( void )`	*Возвращает матрицу преобразования мировой системы координат в систему координат камеры*
`const Matrix3D& GetCameraToWorld ( void )`	*Возвращает матрицу преобразования системы координат камеры в мировую систему координат*
`Vector2D GetScreenScale ( void )`	*Возвращает половинный размер экранной плоскости на единичном расстоянии от наблюдателя в мировой системе координат*
`Vector2D GetPixelSize ( void )`	*Возвращает размер одного пикселя в экранной системе координат при условии, что экранные координаты меняются в диапазоне [-1, 1]*

3. Классы *Mouse* и *Keyboard*

Таблица 2. Поля и методы класса ***Mouse***

Открытые поля	
Синтаксис	*Описание*
`float Step`	*Чувствительность к перемещению мыши*
Конструктор	
Синтаксис	*Описание*
`Mouse ( float step = 0.005F )`	*Создает экземпляр класса для управления камерой при помощи мыши. Параметр step отвечает за чувствительность к перемещению мыши*
Функции обработки событий мыши	

Синтаксис	*Описание*
`void StateChange ( int button, int state )`	*Обрабатывает событие, возникающее при изменении состояния кнопок мыши. В качестве параметров передаются идентификатор кнопки button и ее состояние state от соответствующего обработчика библиотеки GLFW*
`void MouseMove ( int x, int y )`	*Обрабатывает событие, возникающее при перемещении курсора мыши. В качестве параметров передаются координаты x и y курсора мыши от соответствующего обработчика библиотеки GLFW*
Функции применения изменений к камере	
Синтаксис	*Описание*
`void Apply ( Camera * camera )`	*Применяет новые параметры к камере*

Таблица 3. Поля и методы класса ***Keyboard***

Открытые поля	
Синтаксис	*Описание*
`float Step`	*Скорость перемещения камеры*
Конструктор	
Синтаксис	*Описание*
`Keyboard ( float step = 0.005F )`	*Создает экземпляр класса для управления камерой при помощи клавиатуры. Параметр step отвечает за скорость перемещения камеры*
Функции обработки событий клавиатуры	
Синтаксис	*Описание*
`void StateChange ( int key, int state )`	*Обрабатывает событие, возникающее при изменении состояния клавиш клавиатуры:* *W – перемещение вперед* *S – перемещение назад* *A – перемещение влево* *D – перемещение вправо* *X – перемещение вверх* *Z – перемещение вниз* *В качестве параметров передаются идентификатор клавиши key и ее состояние state от соответствующего обработчика библиотеки GLFW*
Функции применения изменений к камере	

Синтаксис	Описание
`void Apply ( Camera * camera )`	Применяет новые параметры к камере

4. Классы `TextureData{1|2|3}D`

Таблица 4. Поля и методы классов `TextureData{1|2|3}D`

Конструкторы	
Синтаксис	Описание

`[1D] TextureData1D ( int width, int components = 3 )`
`[2D] TextureData2D ( int width, int height, int components = 3 )`
`[3D] TextureData3D ( int width, int height, int depth, int components = 3 )`
	Создает новые текстурные данные соответствующей размерности и с заданным числом компонент на тексель

Операторы преобразования типа	
Синтаксис	Описание
`[1D|2D|3D] operator float * ( void )`	Преобразует текстурные данные к массиву элементов типа float
`[1D|2D|3D] operator const float * ( void )`	Преобразует текстурные данные к константному массиву элементов типа float
Функции доступа к текстурным данным	
Синтаксис	Описание

`[1D|2D|3D] template < class Type > Type& Pixel ( int x )`
`[2D] template < class Type > Type& Pixel ( int x, int y )`
`[3D] template < class Type > Type& Pixel ( int x, int y, int z )`
	Обеспечивает поэлементный доступ к текстурным данным, интерпретируя каждый тексель как переменную типа Type. В качестве типа Type могут использоваться следующие типы (в зависимости от числа компонент текселя): float, float *, Vector3D или Vector4D

Функции чтения параметров текстурных данных	
Синтаксис	Описание
`[1D|2D|3D] int GetPixelFormat ( void )`	Возвращает формат представления текселя в OpenGL. Данная функция выбирает формат автоматически в зависимости от числа компонент
`[1D|2D|3D] int GetInternalFormat ( void )`	Возвращает внутренний формат представления текселя в OpenGL. Данная функция выбирает формат автоматически в зависимости от числа компонент
`[1D|2D|3D] int GetType ( void )`	Возвращает тип данных для хранения компонент текселя в OpenGL. В текущей реализации данная функция всегда возвращает GL_FLOAT
`[1D|2D|3D] int GetWidth ( void )`	Возвращает ширину массива

	текстурных данных
`[2D|3D] int GetHeight ( void )`	*Возвращает высоту массива текстурных данных*
`[3D] int GetDepth ( void )`	*Возвращает глубину массива текстурных данных*
`[1D|2D|3D] int GetComponents ( void )`	*Возвращает число компонент текселя*
Функции установки текстурных данных	
Синтаксис	*Описание*

`[1D] void Upload ( int target = GL_TEXTURE_1D )`
`[2D] void Upload ( int target = GL_TEXTURE_2D )`
`[3D] void Upload ( int target = GL_TEXTURE_3D )`
	Передает OpenGL параметры текстурных данных и выгружает их в память графического процессора

5. Классы `FilterMode` и `WrapMode`

Таблица 5. Поля и методы класса `FilterMode`

Открытые поля	
Синтаксис	*Описание*
`int Minification`	*Задает режим фильтрации текстуры при ее уменьшении. Возможные значения: GL_NEAREST и GL_LINEAR*
`int Magnification`	*Задает режим фильтрации текстуры при ее увеличении. Возможные значения: GL_NEAREST и GL_LINEAR*
Конструктор	
Синтаксис	*Описание*
`FilterMode ( int = GL_NEAREST, int = GL_NEAREST )`	*Создает новый объект для задания режимов фильтрации текстуры*
Функции установки параметров фильтрации	
Синтаксис	*Описание*
`void Setup ( int target )`	*Передает заданные параметры фильтрации OpenGL для текстуры типа target. Возможные значения: GL_TEXTURE_{1|2|3}D*

Таблица 6. Поля и методы класса `WrapMode`

Открытые поля	
Синтаксис	*Описание*
`int WrapS`	*Задает режим наложения текстуры для координаты s. Возможные значения: GL_CLAMP и GL_REPEAT*
`int WrapT`	*Задает режим наложения текстуры для координаты t. Возможные*

	значения: GL_CLAMP и GL_REPEAT
`int WrapR`	*Задает режим наложения текстуры для координаты r. Возможные значения: GL_CLAMP и GL_REPEAT*
Конструктор	
Синтаксис	*Описание*

`WrapMode ( int s = GL_CLAMP,`
`int t = GL_CLAMP,`
`int r = GL_CLAMP )`
	Создает новый объект для задания режимов наложения текстуры

Операторы преобразования типа	
Синтаксис	*Описание*
`void Setup ( int target )`	*Передает заданные параметры наложения OpenGL для текстуры типа target. Возможные значения:* *GL_TEXTURE_{1|2|3}D*

6. Классы `Texture{1|2|3}D`

Таблица 7. Поля и методы классов `Texture{1|2|3}D`

Открытые поля	
Синтаксис	*Описание*

`[1D] TextureData1D * Data`
`[2D] TextureData2D * Data`
`[3D] TextureData3D * Data`
	Текстурные данные

`[1D|2D|3D] FilterMode FilterMode`	*Режим фильтрации текстуры*
`[1D|2D|3D] WrapMode WrapMode`	*Режим наложения текстуры*
Конструктор	
Синтаксис	*Описание*

`[1D] Texture1D ( unsigned unit = 0 )`
`[2D] Texture2D ( unsigned unit = 0, unsigned = GL_TEXTURE_2D )`
`[3D] Texture3D ( unsigned unit = 0 )`
	Создает новый текстурный объект, который будет установлен на текстурный модуль unit. Для двумерных текстур можно выбрать конкретный тип текстуры: GL_TEXTURE_2D или GL_TEXTURE_RECTANGLE

`[1D] Texture1D ( TextureData1D * data, unsigned unit = 0 )`
`[2D] Texture2D ( TextureData2D * data, unsigned unit = 0,`
`unsigned = GL_TEXTURE_2D )`
`[3D] Texture3D ( TextureData3D * data, unsigned unit = 0 )`
	Создает новый текстурный объект с заданными данными data, который будет установлен на текстурный модуль unit. Для двумерных текстур можно выбрать конкретный тип текстуры: GL_TEXTURE_2D или GL_TEXTURE_RECTANGLE

Функции управления текстурным объектом	

Синтаксис	Описание
`[1D|2D|3D] void Setup ( void )`	Передает OpenGL параметры текстурного объекта и выгружает данные в память графического адаптера
`[1D|2D|3D] void Update ( void )`	Обновляет данные текстурного объекта в памяти графического адаптера
`[1D|2D|3D] void Bind ( void )`	Присоединяет текстурный объект к состоянию OpenGL
`[1D|2D|3D] void Unbind ( void )`	Отсоединяет текстурный объект от состояния OpenGL
Функции чтения параметров текстурного объекта	
Синтаксис	Описание
`[1D|2D|3D] unsigned GetHandle ( void )`	Возвращает идентификатор текстурного объекта в OpenGL
`[1D|2D|3D] unsigned GetTarget ( void )`	Возвращает тип текстурного объекта в OpenGL
`[1D|2D|3D] unsigned GetUnit ( void )`	Возвращает номер текстурного модуля, на который установлен текстурный объект

7. Классы *RenderBuffer* и *FrameBuffer*

Таблица 8. Поля и методы класса ***RenderBuffer***

Отрытые поля	
Синтаксис	Описание
`unsigned Width`	Ширина буфера визуализации
`unsigned Height`	Высота буфера визуализации
`unsigned Attachment`	Тип буфера визуализации в OpenGL
`unsigned InternalFormat`	Внутренний формат пикселя буфера визуализации в OpenGL
Конструктор	
Синтаксис	Описание

`RenderBuffer ( unsigned int width = 512,`
`unsigned int height = 512,`
`unsigned int attachment = GL_DEPTH_ATTACHMENT,`
`unsigned int format = GL_DEPTH_COMPONENT16 )`
	Создает новый буфер визуализации с заданной шириной width и высотой height типа attachment с внутренним форматом представления элементов format

Функции управления буфером визуализации	
Синтаксис	Описание

`void Setup ( void )`	*Передает OpenGL параметры буфера визуализации и выделяет память для его хранения*
`void Bind ( void )`	*Присоединяет буфер визуализации к состоянию OpenGL*
`void Unbind ( void )`	*Отсоединяет буфер визуализации от состояния OpenGL*
Функции чтения параметров буфера визуализации	
Синтаксис	*Описание*
`int GetHandle ( void )`	*Возвращает идентификатор буфера визуализации в OpenGL*

Таблица 9. Поля и методы класса ***`FrameBuffer`***

Отрытые поля	
Синтаксис	*Описание*
`vector < Texture2D * > ColorBuffers`	*Список двумерных текстур, которые будут использоваться в качестве буферов цвета (не более 8)*
`vector < RenderBuffer * > RenderBuffers`	*Список буферов визуализации, которые будут использоваться в качестве буферов глубины или трафарета*
Конструктор	
Синтаксис	*Описание*
`FrameBuffer ( void )`	*Создает новый буфер кадров, для которого не назначен ни один буфер для вывода*
Функции управления буфером кадров	
Синтаксис	*Описание*
`void Setup ( void )`	*Передает OpenGL параметры буфера кадров и присоединяет к нему все заданные буферы визуализации и двумерные текстуры*
`void Bind ( void )`	*Присоединяет буфер кадров к состоянию OpenGL*
`void Unbind ( void )`	*Отсоединяет буфер кадров от состояния OpenGL*
`void FetchOutcome ( void )`	*Загружает текстурные данные из графической памяти в системную*
Функции чтения параметров буфера кадров	
Синтаксис	*Описание*
`int GetHandle ( void )`	*Возвращает идентификатор буфера кадров в OpenGL*

8. Класс *ShaderManager*

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

Конструктор	
Синтаксис	*Описание*
`ShaderManager( void )`	*Создает новый объект для управления вершинными и фрагментными шейдерными программами*
Функции управления шейдерными программами	
Синтаксис	*Описание*
`bool LoadVertexShader ( const char * filename )`	*Загружает вершинный шейдер из файла filename*
`bool LoadFragmentShader ( const char * filename )`	*Загружает фрагментный шейдер из файла filename*
`bool LoadVertexShader ( const char ** filenames, int count )`	*Загружает вершинный шейдер из count файлов с именами в массиве filenames*
`bool LoadFragmentShader ( const char ** filenames, int count )`	*Загружает фрагментный шейдер из count файлов с именами в массиве filenames*
`void BuildProgram ( void )`	*Собирает шейдеры в единую программу для исполнения на графическом процессоре*
`void Bind ( void )`	*Присоединяет шейдерную программу к состоянию OpenGL*
`void Unbind ( void )`	*Отсоединяет шейдерную программу от состояния OpenGL*
Функции передачи данных в шейдеры	
Синтаксис	*Описание*
`int GetUniformLocation ( char * name )`	*Возвращает идентификатор uniform-переменной с именем name в OpenGL*
`int GetAttributeLocation ( char * name )`	*Возвращает идентификатор attribute-переменной с именем name в OpenGL*
`Vector4D GetUniformVector ( char * name )`	*Возвращает значение uniform-вектора с именем name в OpenGL*
`Vector4D GetUniformVector ( int location )`	*Возвращает значение uniform-вектора с идентификатором location в OpenGL*
`Vector4D GetAttributeVector ( char * name )`	*Возвращает значение attribute-вектора с именем name в OpenGL*
`Vector4D GetAttributeVector ( int location )`	*Возвращает значение attribute-вектора с идентификатором location в OpenGL*
`bool SetUniformInteger ( const char * name, int value )`	*Устанавливает значение value целочисленной uniform-переменной с именем name в OpenGL*
`bool SetUniformInteger ( int location, int value )`	*Устанавливает значение value целочисленной uniform-переменной с идентификатором location в OpenGL*

`bool SetUniformFloat ( const char * name, float value )`	Устанавливает значение value вещественной uniform-переменной с именем name в OpenGL
`bool SetUniformFloat ( int location, float value )`	Устанавливает значение value вещественной uniform-переменной с идентификатором location в OpenGL
`bool SetUniformVector ( const char * name, const Vector2D& value )`	Устанавливает значение value двумерного uniform-вектора с именем name в OpenGL
`bool SetUniformVector ( int location, const Vector2D& value )`	Устанавливает значение value двумерного uniform-вектора с идентификатором location в OpenGL
`bool SetUniformVector ( const char * name, const Vector3D& value )`	Устанавливает значение value трехмерного uniform-вектора с именем name в OpenGL
`bool SetUniformVector ( int location, const Vector3D& value )`	Устанавливает значение value трехмерного uniform-вектора с идентификатором location в OpenGL
`bool SetUniformVector ( const char * name, const Vector4D& value )`	Устанавливает значение value четырехмерного uniform-вектора с именем name в OpenGL
`bool SetUniformVector ( int location, const Vector4D& value )`	Устанавливает значение value четырехмерного uniform-вектора с идентификатором location в OpenGL
`bool SetUniformMatrix ( const char * name, const Matrix2D& value )`	Устанавливает значение value двумерной uniform-матрицы с именем name в OpenGL
`bool SetUniformMatrix ( int location, const Matrix2D& value )`	Устанавливает значение value двумерной uniform-матрицы с идентификатором location в OpenGL
`bool SetUniformMatrix ( const char * name, const Matrix3D& value )`	Устанавливает значение value трехмерной uniform-матрицы с именем name в OpenGL
`bool SetUniformMatrix ( int location, const Matrix3D& value )`	Устанавливает значение value трехмерной uniform-матрицы с идентификатором location в OpenGL
`bool SetUniformMatrix ( const char * name, const Matrix4D& value )`	Устанавливает значение value четырехмерной uniform-матрицы с именем name в OpenGL
`bool SetUniformMatrix ( int location, const Matrix4D& value )`	Устанавливает значение value четырехмерной uniform-матрицы с идентификатором location в OpenGL
`bool SetTexture ( int location, const Texture1D * texture )`	Устанавливает одномерный текстурный объект texture с идентификатором location в OpenGL
`bool SetTexture ( const char * name, const Texture1D * texture )`	Устанавливает одномерный текстурный объект texture с именем name в OpenGL
`bool SetTexture ( int location, const Texture2D * texture )`	Устанавливает двумерный текстурный объект texture с идентификатором location в OpenGL
`bool SetTexture ( const char * name, const Texture2D * texture )`	Устанавливает двумерный текстурный объект texture с именем

	name в OpenGL
`bool SetTexture ( int location, const Texture3D * texture )`	*Устанавливает трехмерный текстурный объект texture с идентификатором location в OpenGL*
`bool SetTexture ( const char * name, const Texture3D * texture )`	*Устанавливает трехмерный текстурный объект texture с именем name в OpenGL*
`bool SetAttributeName ( int location, const char * name )`	*Назначает имя name attribute-переменной с идентификатором location*
`bool SetAttributeFloat ( const char * name, float value )`	*Устанавливает значение value вещественной attribute-переменной с именем name в OpenGL*
`bool SetAttributeFloat ( int location, float value )`	*Устанавливает значение value вещественной attribute-переменной с идентификатором location в OpenGL*
`bool SetAttributeVector ( const char * name, const Vector2D& value )`	*Устанавливает значение value двумерного attribute-вектора с именем name в OpenGL*
`bool SetAttributeVector ( int location, const Vector2D& value )`	*Устанавливает значение value двумерного attribute-вектора с идентификатором location в OpenGL*
`bool SetAttributeVector ( const char * name, const Vector3D& value )`	*Устанавливает значение value трехмерного attribute-вектора с именем name в OpenGL*
`bool SetAttributeVector ( int location, const Vector3D& value )`	*Устанавливает значение value трехмерного attribute-вектора с идентификатором location в OpenGL*
`bool SetAttributeVector ( const char * name, const Vector4D& value )`	*Устанавливает значение value четырехмерного attribute-вектора с именем name в OpenGL*
`bool SetAttributeVector ( int location, const Vector4D& value )`	*Устанавливает значение value четырехмерного attribute-вектора с идентификатором location в OpenGL*