**sxgcore** – графическое ядро движка SkyXEngine, использует технологии DirectX 9.

**Базовые функции sxgcore**

long SGCore\_0GetVersion();

версия подсистемы

void SGCore\_Dbg\_Set(report\_func rf);

установка функции вывода сообщений

void SGCore\_0Create(const char\* name, HWND hwnd, int width, int heigth, bool windowed, DWORD create\_device\_flags, bool is\_unic = true);

инициализация графического ядра, параметры:

* name - передаваемое имя подсистемы
* hwnd - дескриптор окна, в которое будет осуществляться рендер
* width - ширина области рендера
* heigth - высота области рендера
* windowed - оконный режим использовать ли? иначе полноэкранный
* create\_device\_flags - флаги создания устройства (стандартные dx)
* is\_unic - должна ли подсистема быть уникальной на основе имени

void SGCore\_0Kill();

уничтожение либы

IDirect3DDevice9\* SGCore\_GetDXDevice();

возвращает dx устройство

void SGCore\_ScreenQuadDraw();

отрисовка full screen quad (уже смещенного как надо чтобы не было размытия)

**Базовые функции установки настроек выборок из текстур:**

Установка фильтрации (value - D3DTEXTUREFILTERTYPE):

void SGCore\_SetSamplerFilter(DWORD id, DWORD value);

для одного слота id

void SGCore\_SetSamplerFilter2(DWORD begin\_id, DWORD end\_id, DWORD value);

для набора слотов от begin\_id до end\_id

Установка адресации:

void SGCore\_SetSamplerAddress(DWORD id, DWORD value);

для одного слота id

void SGCore\_SetSamplerAddress2(DWORD begin\_id, DWORD end\_id, DWORD value);

для набора слотов от begin\_id до end\_id

**Обработка потери/восстановления устройства**

***Примечание****: функции обработки потери/восстановления устройства обрабатывают ресурсы только в пределах данной библиотеки, другие библиотеки должны сами производить данную обработку!*

void SGCore\_OnLostDevice();

сбрасывает все зависимые ресурсы, вызывать при потере устройства

bool SGCore\_OnDeviceReset(int width,int heigth,bool windewed);

сброс устройства и установка заданных ему параметров, возвращает статус попытки, вызывать при попытке сброса устройства

void SGCore\_OnResetDevice();

восстановление зависимых ресурсов, вызывать при сбросе устройства

**Отладочное сообщение в окне рендера**

Графическое ядро предусматривает наличие текстового сообщения в окне рендера, которое формирует непосредственно сам программист и дает команду на его вывод посредством вызова функции:

void SGCore\_DbgMsg(const char\* format, ...);

Аргументы аналогичны стандартным функциям типа printf.

Размер отладочного сообщения, выводимого в окно рендера ограничено макроопределением SXGC\_STR\_SIZE\_DBG\_MSG.

**Базовые переопределяемые функции**

sxgcore содержит базовые и необходимые функции для переопределения их пользователем, которые будут доступны из графического ядра в других библиотеках зависимых от графического ядра, но в тоже время, эти функции могут быть переопределены функциями из других библиотек. Это обеспечивает централизованную обобщенную зависимость от самого графического ядра и исключает перекрестные зависимости библиотек между собой, позволяя программисту воздействовать на ценрт управления не из центра.

Переопределяемые функции могут быть переопределены во внутреннем состоянии, то есть функция обертка так и останется функцией оберткой только будет вызывать уже переопределенную функцию.

Переопределяемые функции изначально выполняют штатные действия исходя из их назначения, и могут вообще не переопределяться если так надо программисту, то есть необходимость в их переопределении исходит только от программиста.

**Переопределяемые функции:**

void SGCore\_DIP(UINT type\_primitive, long base\_vertexIndex, UINT min\_vertex\_index, UINT num\_vertices, UINT start\_index, UINT prim\_count);

аналогична DrawIndexedPrimitive, в дополнение к DIP инкрементирует счетчик DIPов в int регистрах по индексу G\_RI\_INT\_COUNT\_DIP, обнуление данных только на стороне приложения

void SGCore\_MtlSet(ID id, float4x4\* world);

установка материала (по умолчанию - текстуры в нулевой текстурный слот), id – идентификатор материала (по умолчанию - текстуры), world – матрица трансформации модели

ID SGCore\_MtlLoad(const char\* name, int mtl\_type)

загрузка материала (по умолчанию – текстуры) с именем name, mtl\_type – типа материала, будет задействован только в случае провала определения типа материала (по умолчанию не используется) - тип стандартного материала будет определен на основании mtl\_type, может принимать одно из следующих значений:

#define MTL\_TYPE\_GEOM 0

#define MTL\_TYPE\_GRASS 1

#define MTL\_TYPE\_TREE 2

#define MTL\_TYPE\_SKIN 3

int SGCore\_MtlGetSort(ID id);

получить «сорт» материала (по умолчанию 0)

bool SGCore\_MtlGroupRenderIsSingly(ID id);

возвращает true если подгруппа требует индивидуального рендера (к примеру для отражений), иначе false (по умолчанию)

**Прототипы переопределяемых функций:**

typedef void(\*g\_func\_dip) (UINT type\_primitive, long base\_vertexIndex, UINT min\_vertex\_index, UINT num\_vertices, UINT start\_index, UINT prim\_count);

typedef void(\*g\_func\_mtl\_set) (ID id, float4x4\* world);

typedef ID(\*g\_func\_mtl\_load) (const char\* name, int mtl\_type);

typedef int(\*g\_func\_mtl\_get\_sort) (ID id);

typedef bool(\*g\_func\_mtl\_group\_render\_is\_singly) (ID id);

**Переопределение функций (установка новых функций для вызова):**

void SGCore\_SetFunc\_DIP(g\_func\_dip func);

void SGCore\_SetFunc\_MtlSet(g\_func\_mtl\_set func);

void SGCore\_SetFunc\_MtlLoad(g\_func\_mtl\_load func);

void SGCore\_SetFunc\_MtlGetSort(g\_func\_mtl\_get\_sort func);

void SGCore\_SetFunc\_MtlGroupRenderIsSingly(g\_func\_mtl\_group\_render\_is\_singly func);

**Шейдеры (shaders)**

sxgcore поддерживает HLSL шейдеры версии 3.0. Создание и использование шейдеров происходит посредством функций предоставляемых sxgcore, непосредственного интерфейса нет.

Поддерживаются следующие типы шейдеров:

enum ShaderType

{

st\_vertex, //вершинный

st\_pixel //пиксельный

};

Расширения шейдеров:

* vs – вершинный
* ps - пиксельный

Распознавание (идентификация) шейдеров (обращение через функции) происходит на основе идентификаторов представленных типом ID, данный идентификатор присваивается при загрузке шейдера и взвращается загружаемой функцией пользователю. Если речь идет об идентификации по имени то имя в данном случае эта та строка name которая была передана SGCore\_ShaderLoad.

Также реализация системы шейдеров предусматривает устранение дубликатов шейдеров с одинаковыми данными, данные в здесь это:

* пользовательское имя шейдера, присвоено при его загрузке самим пользователем
* имя файла шейдера

Перечисление ShaderCheckDouble позволяет установить каким образом будет произведен поиск загружаемого шейдера на наличие его экземпляра в списке уже загруженных:

enum ShaderCheckDouble

{

scd\_none, //нет проверки

scd\_path, //проверка по пути (имени шейдера с расширением)

scd\_name //проверка по пользовательскому имени

};

Если имя шейдера не содержит нижний пробел \_ то значит шейдер находится в корне директории. Если имя шейдера содержит нижний пробел \_ то строка до первого нижнего проблема это имя папки в котором находится шейдер с целым именем pp\_shader.vs - лежит по загружаемому пути: /pp/pp\_shader.vs.

Также существуют ограничения на длину имени шейдера и составных частей имени шейдера:

SXGC\_SHADER\_MAX\_SIZE\_DIR максимальный размер имени директории (до \_)

SXGC\_SHADER\_MAX\_SIZE\_NAME максимальный размер имени с расширением (после \_)

Длина имени переменной в коде шейдера имеет ограничение SXGC\_SHADER\_VAR\_MAX\_SIZE.

Максимальное количество uniform переменных ограничено SXGC\_SHADER\_VAR\_MAX\_COUNT.

Количество уникальных имен макроопределений внутри шейдера ограничивается SXGC\_SHADER\_COUNT\_MACRO.

Перед использованием функция для работы с шейдерами необходимо инициализировать путь относительно которого будут браться шейдеры:

void SGCore\_ShaderSetStdPath(const char\* path);

установить абсолютный путь откуда брать шейдеры

void SGCore\_ShaderGetStdPath(char\* path);

возвращает абсолютный путь откуда берутся шейдеры

**Функции для работы с шейдерами:**

ID SGCore\_ShaderLoad(ShaderType type\_shader, const char\* path, const char\* name, ShaderCheckDouble is\_check\_double, D3DXMACRO\* macro = 0);

загрузка шейдера, возвращает числовой идентификатор (порядковый номер) по которому доступен загруженный шейдер, параметры:

* type\_shader - тип шейдера из ShaderType
* path - имя файла шейдера с расширением
* name - имя шейдера которое присвоится при загрузке
* is\_check\_double - проверка ли на уникальность из ShaderCheckDouble
* macro – макросы, которые будут переданы в шейдер, указав 0 данные не будут переданы

void SGCore\_ShaderGetName(ShaderType type\_shader, ID id, char\* name);

записывает пользовательское имя шейдера с типом type\_shader и идентификатором который был выдан при загрузке id в name

ID SGCore\_ShaderIsExistName(ShaderType type\_shader, const char\* name);

существует ли шейдер с типом type\_shader и пользовательским именем name, если да то возвращает id, иначе < 0

ID SGCore\_ShaderIsExistPath(ShaderType type\_shader, const char\* path);

существует ли шейдер с типом type\_shader и именем файла и расширением name, если да то возвращает id, иначе < 0

bool SGCore\_ShaderIsValidate(ShaderType type\_shader, ID id);

загружен ли шейдер с типом type\_shader и с идентификатором id

void SGCore\_ShaderReloadAll()

перезагрузить все шейдеры, с учетом макросов

void SGCore\_ShaderUpdate(ShaderType type\_shader, ID id, D3DXMACRO macro[] = 0);

обновление шейдера с типом type\_shader и идентификатором id с передачей массива макроопределений macro

void SGCore\_ShaderUpdateN(ShaderType type\_shader, const char\* name, D3DXMACRO macro[] = 0)

обновление шейдера с типом type\_shader и именем name с передачей массива макроопределений macro

ID SGCore\_ShaderGetID(ShaderType type\_shader, const char\* name);

получить идентификатор шейдера с типом type\_shader по имени name

void SGCore\_ShaderBindN(ShaderType type\_shader, const char\* name);

установка типа шейдера type\_shader с именем name

void SGCore\_ShaderBind(ShaderType type\_shader, ID id);

установка типа шейдера type\_shader с идентификатором id

void SGCore\_ShaderUnBind();

обнуление установок всех типов шейдеров

void SGCore\_ShaderSetVRFN(ShaderType type\_shader, const char\* name\_shader, const char\* name\_var, void\* data, int count\_float4 = 0)

передача float данных uniform переменным в шейдер с именем, параметры:

* type\_shader – тип шейдера из ShaderType
* name\_shader – пользовательское имя шейдера присвоенное при его загрузке
* name\_var – имя переменной, которой будут присвоенные данные
* data – указатель на float массив
* count\_float4 – количество float4 в передаваемом массиве данных, минимальное значение 1, данный параметр можно не указывать и тогда данные о размере будут взяты из шейдера

void SGCore\_ShaderSetVRF(ShaderType type\_shader, ID id, const char\* name\_var, void\* data, int count\_float4 = 0);

передача float данных uniform переменным в шейдер с идентификатором id, параметры см. SGCore\_ShaderSetVRFN

void SGCore\_ShaderSetVRIN(ShaderType type\_shader, const char\* name\_shader, const char\* name\_var, void\* data, int count\_int4 = 0);

передача int данных uniform переменным в шейдер с именем, параметры:

* type\_shader – тип шейдера из ShaderType
* name\_shader – пользовательское имя шейдера присвоенное при его загрузке
* name\_var – имя переменной, которой будут присвоенные данные
* data – указатель на int массив
* count\_int4 – количество int4 в передаваемом массиве данных, минимальное значение 1, данный параметр можно не указывать и тогда данные о размере будут взяты из шейдера

void SGCore\_ShaderSetVRI(ShaderType type\_shader, ID id, const char\* name\_var, void\* data, int count\_int4 = 0);

передача float данных uniform переменным в шейдер с идентификатором id, параметры см. SGCore\_ShaderSetVRIN

**Загружаемые текстуры**

sxgcore предоставляет необходимый функционал для загрузки и работы с загружаемыми текстурами, стандартными средствами DirectX 9.

Распознавание (идентификация) текстур (обращение через функции) происходит на основе идентификаторов представленных типом ID, данный идентификатор присваивается при добавлении текстуры в очередь загрузки и возвращается загружаемой функцией пользователю.

Все идентификаторы являются константными и не могут меняться во внутренней реализации, это значит, что текстура имеющая идентификатор будет доступна в любой момент времени работы по этому же идентификатору, за исключением случаев удаления текстуры.

Имя текстуры обязательно должно содержать нижний пробел \_, строка до первого нижнего проблема это имя папки в котором находится текстура с целым именем, пример: mtl\_tex.dds - лежит по загружаемому пути: /mtl/mtl\_tex.dds

Имя текстуры и его составляющие имеют ограничения по длине:

SXGC\_LOADTEX\_MAX\_SIZE\_DIR - максимальный размер имени директории (до \_)

SXGC\_LOADTEX\_MAX\_SIZE\_NAME - максимальный размер имени с расширением (после \_)

Все загружаемые текстуры после загрузки имеют количество mipmap уровней определеяемое SXGC\_LOADTEX\_COUNT\_MIPMAP.

Макроопределение SXGC\_LOADTEX\_FORMAT\_TEX устанавливает формат всех загружаемых текстур.

Кроме загрузки текстур также есть возможность добавления созданных текстур, однако такие текстуры должны быть созданы в managed pool (D3DPOOL\_MANAGED) ибо обработка потери/восстановления устройства не вызывается для загружаемых (созданных и добавленных) текстур.

Все загруженные/созданные текстуры имеют внутреннюю маркировку:

enum LoadTexType

{

ltt\_load, //загружаемая

ltt\_const, //неудаляемая загружаемая

ltt\_custom, //созданная пользователем

//самоопределение типа, на тот случай когда мы обновляем текстуру которая точно есть

//если определить этот тип, а внутри у текстуры на самом деле нет типа (скорее всего нет текстуры)

//то будет определен ltt\_load

ltt\_self,

};

Перед использованием функция для работы с загружаемыми текстурами необходимо установить путь относительно которого будут браться текстуры:

void SGCore\_LoadTexStdPath(const char\* path);

установить стандартный путь откуда брать текстуры

**Функции для работы с загружаемыми текстурами:**

ID SGCore\_LoadTexAddName(const char\* name, LoadTexType type);

добавить имя текстуры в очередь загрузки, возвращает ID данной текстуры (если была добавлена либо загружена то дубликат не будет создан, вместо этого будет использован ID уже загруженной/поставленной в очередь текстуры)

void SGCore\_LoadTexDelete(ID id);

удалить тектуру по id (независимо от типа)

void SGCore\_LoadTexClearLoaded();

очистить список загружаемых текстур имеющих маркер LoadTexType::ltt\_load

ID SGCore\_LoadTexGetID(const char\* name);

получить id по имени если текстура загружена либо поставлена в очередь, иначе <0

ID SGCore\_LoadTexCreate(const char\* name, IDirect3DTexture9\* tex);

создать место для текстуры tex и присвоить ей имя name, возвращает id, имя должно соответствовать требованиям имен обычных загружаемых текстур

ID SGCore\_LoadTexUpdateN(const char\* name, LoadTexType type);

обновить/перезагрузить текстуру name, если текстуры не было в списке то добавляет, если текстуру надо обновить, но тип у нее заранее не известен, но она точно уже загружена, то можно использовать тип самоопределения ltt\_self, тогда тип текстуры не изменится

void SGCore\_LoadTexUpdate(ID id);

обновить/перезагрузить текстуру

void SGCore\_LoadTexLoadTextures();

загрузка всех текстур поставленных в очередь, если есть очередь

IDirect3DTexture9\* SGCore\_LoadTexGetTex(ID id);

возвращает текстуру по id если такая текстура есть, иначе 0

**Render targets (цели визуализации)**

sxgcore предоставляет набор функция для удобной работы с render target’ами (сокращенно rt).

Распознавание (идентификация) rt (обращение через функции) происходит на основе идентификаторов представленных типом ID, данный идентификатор присваивается при создании нового rt и возвращается загружаемой функцией пользователю. Если речь идет об идентификации по имени, то имя в данном случае эта та строка name которая была передана SGCore\_RTAdd.

Все идентификаторы являются константными и не могут меняться во внутренней реализации, это значит, что rt имеющий идентификатор будет доступен в любой момент времени работы по этому же идентификатору, за исключением случаев удаления rt.

**Функции для работы с render target:**

ID SGCore\_RTAdd(UINT width, height, UINT levels,DWORD usage, D3DFORMAT format, D3DPOOL pool, const char\* name, float coeffullscreen);

добавить новый render target и возвращает ID для доступа к нему, параметры:

* width - ширина
* height - высота
* levels - количество mip-map уровней
* usage -
* format - формат из D3DFORMAT
* pool - где будет размещена текстура, из D3DPOOL
* name-имя rt
* coeffullscreen - коэффициент размеров rt относительно области рендера, (если rt и размер области одинаковы то 1, если rt меньше на 0.5 то 0.5), если указаны фиксированные значения то ставить 0, это нужно для установления размеров при восстановлении устройства

void SGCore\_RTDeleteN(const char\* name);

удалить rt по имени name

void SGCore\_RTDelete(ID id);

удалить rt по id

ID SGCore\_RTGetNum(const char\* name);

возвращает идентификатор по имени name

IDirect3DTexture9\* SGCore\_RTGetTextureN(const char\* name);

возвращает текстуру по имени name

IDirect3DTexture9\* SGCore\_RTGetTexture(ID id);

возвращает текстуру по id

**Статическая модель dse формата**

sxcore предоставляет возможность загрузки статических моделей, формат вершин которых представлен структурой vertex\_static, которая объявлена в файле ModelFile.h:

struct vertex\_static: public model\_vertex

{

float3\_t Pos;

float2\_t Tex;

float3\_t Norm;

};

Итоговая загруженная модель представлена структурой ISXDataStaticModel:

struct ISXDataStaticModel : public IBaseObject

{

virtual ~ISXDataStaticModel(){};

virtual ISXDataStaticModel\* GetCopy()=0; //получить абсолютную копию модели

IDirect3DVertexBuffer9\* VertexBuffer; //вершинный буфер

IDirect3DIndexBuffer9\* IndexBuffer; //индексный буфер

UINT SubsetCount; //количество подгрупп

char\*\* ArrTextures; //массив имен текстур без расширения

UINT\* StartIndex; //массив стартовых позиций индексов для каждой подгруппы

UINT\* IndexCount; //массив количества индексов для каждой подгруппы

UINT\* StartVertex; //массив стартовых позиций вершин для каждой подгруппы

UINT\* VertexCount; //массив количества вершин для каждой подгруппы

UINT AllIndexCount; //общее количество индексов

UINT AllVertexCount;//общее количество вершин

};

**Функции для работы со статическими меодялми dse формата:**

IDirect3DVertexDeclaration9\* SGCore\_StaticModelGetDecl();

возвращает декларацию вершин статической модели

ISXDataStaticModel\* SGCore\_StaticModelCr();

создать статическую модель

void SGCore\_StaticModelLoad(const char\* file, ISXDataStaticModel\*\* data);

загрузить статическую модель, file – абсолютный путь, data инициализируется внутри

void SGCore\_StaticModelSave(const char\* file, ISXDataStaticModel\*\* data);

сохранить статическую модель, file – абсолютный путь

**Ограничивающий объем**

sxgcore предусматривает возможность описания объектов более простыми геометрическими фигурами, такими как сфера и параллелепипед (точнее точки экстремума – минимум и максимум). Также есть возможность их трансформации.

**Базовый класс**

Базовым классом является ISXTransObject, который содержит необходимые элементы описания трансформаций позиции (Position), вращения (Rotation), масштабирования (Scale):

struct ISXTransObject : public IBaseObject

{

virtual ~ISXTransObject(){};

SX\_ALIGNED\_OP\_MEM

virtual inline float4x4\* CalcWorld() = 0;

float3 Position;

float3 Rotation;

float3 Scale;

float4x4 World;

};

Для получения мировой матрицы трансформации необходимо соответствующим векторам присвоить нужные значения для трансформаций, после чего вызывать CalcWorld(), данная функция произведет все трансформации и сформирует мировую матрицу трансформаций для объекта в World и вернет указатель на нее.

Для создания ISXTransObject нужно воспользоваться функцией:

ISXTransObject\* SGCore\_CrTransObject();

**Ограничивающий объем в пространстве экрана SXPosBBScreen**

Структура, описывающая ограничивающий квадрат в пространстве экрана

struct SXPosBBScreen

{

float x; //позиция по оси x в пространстве экрана

float y; //позиция по оси y в пространстве экрана

float width; //ширина в пикселях

float height; //высота в пикселях

float maxdepth; //конец объекта глубина 0-1

float mindepth; //начало объект глубина 0-1

bool IsVisible; //виден ли квадрат наблюдателю

};

Возможно использование для Occulusion culling.

**Интерфейс ограничивающего объема ISXBound**

Создание экземпляра ISXBound:

ISXBound\* SGCore\_CrBound();

**ISXBound состав:**

|  |  |
| --- | --- |
| void CalcBound(IDirect3DVertexBuffer9\* vertex\_buffer, DWORD count\_vert, DWORD bytepervert) | Просчет ограничивающего объема по вершинному буферу vertex\_buffer, в вершинах которого первым элементом идет позиция float3 вектор, count\_vert – количество вершин, bytepervert – количество байт в вершине |
| float4x4\* CalcWorldAndTrans() | функция просчета мировой матрицы и трансформации минимума и максимума, а также сферы, возвращает указатель на внутреннюю мировую матрицу |
| void GetPosBBScreen(SXPosBBScreen \*res, float3\* campos, float3\* sizemapdepth, float4x4\* mat) | Просчитает и запишет в res структуру SXPosBBScreen (ограничивающий объем в пространстве экрана), campos – позиция наблюдателя, sizemapdepth - , mat – произведение видовой и проекционной матриц |
| void SetMinMax(float3\* min, float3\* max) | установить экстремум, также просчитает и сферу |
| void GetMinMax(float3\* min, float3\* max) | запишет в min и max точки экстремума |
| void SetSphere(float3\* center, float\* radius) | установить сферу, просчитает также и параллелепипед |
| void GetSphere(float3\* center, float\* radius) | запишет в center центр сферы, в radius радиус сферы |
| bool IsPointInSphere(float3\* point) | находится ли точка point в пределах сферы |
| bool IsPointInBox(float3\* point) | находится ли точка point в пределах параллелепипеда |

**Камера, frustum (SXFrustumPlane, ISXFrustum, ISXCamera)**

**Frustum**

Frustum представлен 6 плоскостями, которые описывают всю ту область которую видит камера.

Плоскости в sxgcore представлены структурой SXFrustumPlane.

Frustum в sxgcore представлен ISXFrustum.

Функция создания:

ISXFrustum\* SGCore\_CrFrustum();

**ISXFrustum состав:**

|  |  |
| --- | --- |
| void Update(const float4x4\* view, const float4x4\* proj) | обновление плоскостей фрустума, на вход идут видовая и проекционная матрицы |
| bool PointInFrustum(const float3 \*point) | находится ли точка во фрустуме |
| bool PolyInFrustum(const float3\* p1, const float3\* p2, const float3\* p3) | находится ли треугольник во врутстуме |
| bool PolyInFrustumAbs(const float3\* p1, const float3\* p2, const float3\* p3) | находится ли полигон во фрустуме полностью |
| bool SphereInFrustum(const float3 \*point, float radius) | находится ли сфера во фрустуму |
| bool SphereInFrustumAbs(const float3 \*point, float radius) | находится ли сфера во фрустуму полностью |
| bool BoxInFrustum(float3\* min, float3\* max) | находится ли параллелепипед (описанный точками экстремума) во врустуме |

Камера