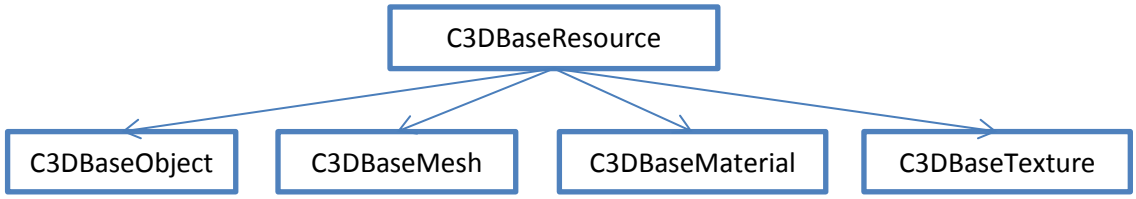
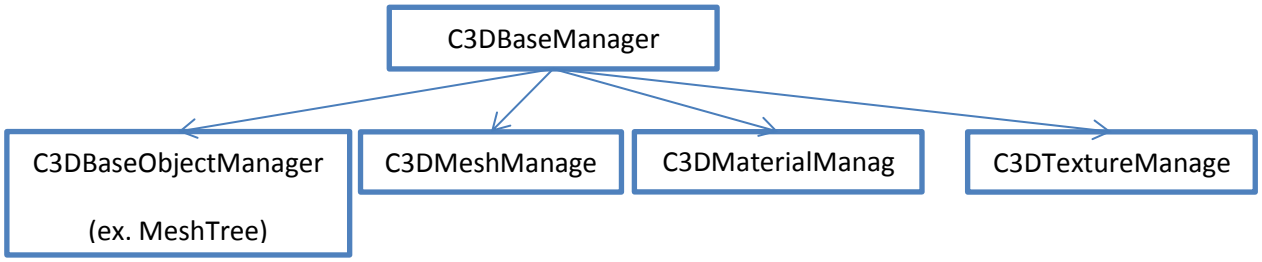
Описание концепции ResourceManager

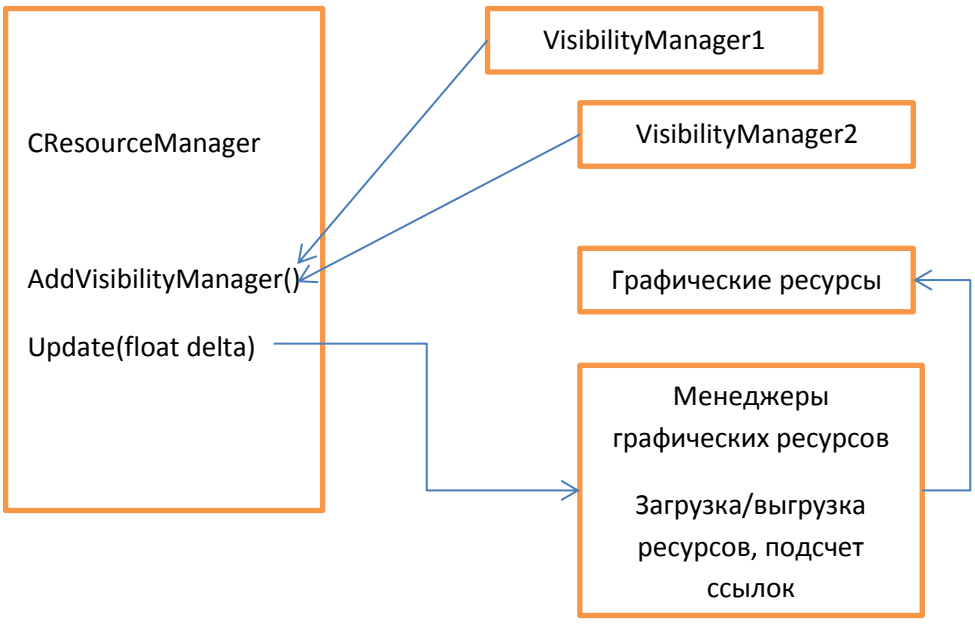
Функциональность ResourceManager, по аналогии с VisibilityManager предоставляется отдельной динамической библиотекой, экспортирующей класс CResourceManager. CResourceManager может взаимодействовать одновременно с несколькими объектами CVisibilityManager, в целях поддержки многоканальной визуализации. Функции CResourceManager заключаются в определении необходимости асинхронной загрузки/выгрузки графических ресурсов.

Графические ресурсы и управляющие классы представляются следующей иерархией наследственности:

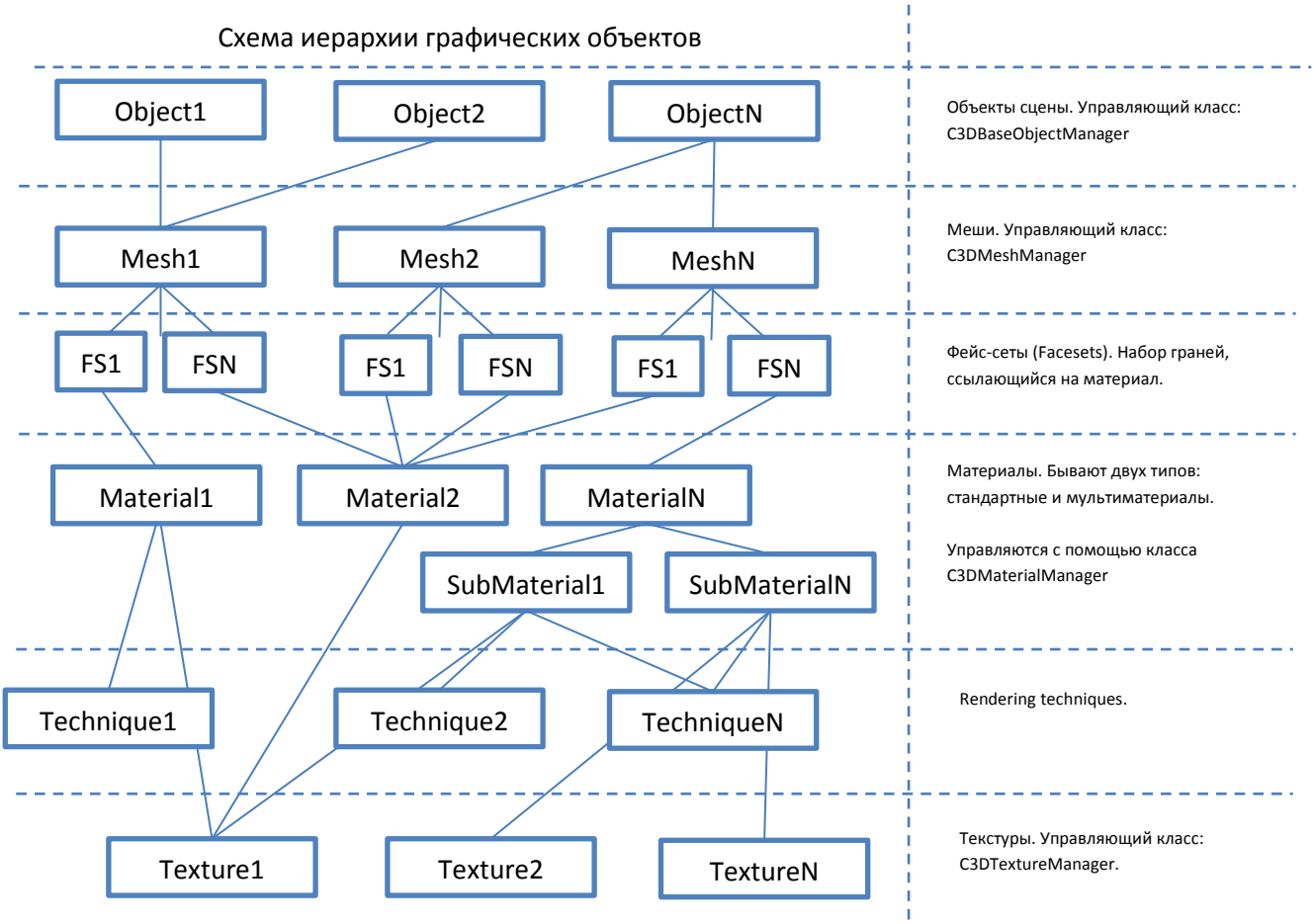




На следующей схеме изображен механизм взаимодействия CResourceManager с пользователем и VisibilityManager:



Графические ресурсы сцены имеют следующую иерархию:



Структуры для взаимодействия с ResourceManager:

```
enum E3DResourceType
{
	C3DRESOURCE_OBJECT,
	C3DRESOURCE_MESH,
	C3DRESOURCE_MATERIAL,
	C3DRESOURCE_FACESET,
	C3DRESOURCE_TECHNIQUE,
	C3DRESOURCE_TEXTURE
};

// базовый ресурс
struct C3DBaseResource
{
	E3DResourceType	GetType() const = 0;

	// получить количество ссылок видимых рсурсов на данный ресурс
	virtual size_t GetVisibleRefCount() const { return _visibleRefCount; }

	// получить родительские ресурсы
	virtual void GetParentResources(std::vector<C3DBaseResource*>&
out_vecParentResources) const = 0;

	// получить дочерние ресурсы
```

```
	virtual void GetChildResources(std::vector<C3DBaseResource*>&
out_vecChildResources) const = 0;

	// получить указатель на менеджер, управляющий данным ресурсом
	virtual C3DBaseManager*	GetManager() const = 0;

private:

	size_t _visibleRefCount;

	void AddVisibleRef();
	void ReleaseVisibleRef();

	friend class CResourceManager;
};

// базовый менеджер графических ресурсов
struct C3DBaseManager
{
	// Запросить загрузку ресурса
	virtual void RequestLoadResource(C3DBaseResource*) = 0;

	// запросить выгрузку ресурса
	virtual void RequestUnloadResource(C3DBaseResource*) = 0;
};

// базовый класс менеджера объектов
struct C3DBaseObjectManager : public C3DBaseManager
{
	// получить список объектов по заданному баунд-боксу
	virtual void GetObjectList(const D3DXVECTOR3& bboxMin, const D3DXVECTOR3& bboxMax,
std::vector<C3DBase3DObject*>& out_vecObjects) = 0;
};

// базовый класс менеджера объекта в сцене
struct C3DBaseObject : public C3DBaseResource
{
	// Все 3D объекты должны будут возвращать Баунд-Бокс. Причем, если объект - точка,
а не меш, то
	// пусть вернет одинаковые значения в out_vBBMin и out_vBBMax.
	virtual void GetBoundBox(D3DXVECTOR3** ppBBMin, D3DXVECTOR3** ppBBMax) = 0;

	// Получить матрицу трансформации
	virtual D3DXMATRIX* GetWorldTransform() = 0;

	// Функция должна возвращать: включена-ли проверка размера объекта на экране
	virtual bool IsMinimalSizeCheckEnabled() const = 0;
};

// базовый класс меша
struct C3DBaseMesh : public C3DBaseResource
{
	// Получить список Фейс-Сетов, используемых в меше
	virtual void  GetFaceSets(vector<C3DBaseFaceSet*>& out_vecFaceSets) const = 0;
};

// базовый класс фейс-сета
struct C3DBaseFaceSet : public C3DBaseResource
{
	// получить родительский меш
	virtual C3DBaseMesh* GetParentMesh() = 0;

	// получить ссылку на материал
	virtual C3DBaseMaterial* GetMaterialRef() = 0;
};
```

```
// базовый материал
struct C3DBaseMaterial : public C3DBaseResource
{
	// Функция используется в процессе рендеринга [для взаимодействия процесса рендера
с Visman]
	virtual void  AddVisibleFaceSet(C3DBaseFaceSet*) = 0;

	// Получить набор техник
	virtual void  GetTechniques(std::vector<C3DBaseTechnique*>& out_vecTechniques)
const = 0;

	// получить список текстур
	virtual void  GetTextures(std::vector<C3DBaseTexture*>& out_vecTextures) const = 0;
};

struct C3DBaseTechnique : public C3DBaseResource
{

};

enum ETextureType
{
	eTextureType_2D = 0,
	eTextureType_CubeMap,
	eTextureType_Volume,
	eTextureType_FX,

	eTextureTypeCount
};


struct C3DBaseTexture : public C3DBaseResource
{
	virtual ETextureType GetTextureType() const = 0;
};

class CResourceManager
{
public:

	// инициализировать, указав MeshTree
	void Init(C3DBaseObjectManager*);
	// указать время, спустя которое, текстура став не видимой
	// для всех VisibilityManager выгрузится из памяти
	// [настроечный параметр, по умолчанию = 25с]
	void SetInvisibleUnloadTime(float time);

	// добавить VisibilityManager в обработку
	// [вызывается во время инициализации]
	void AddVisibilityManager(CVisibilityManager*);

	// Обновить состояние ресурсов
	// [можно вызывать в отдельном потоке, не привязываясь к циклу рендера]
	void Update(float deltaTime);

	// Получить текущий приоритет текстуры при отрисовке
	float GetTexturePriority(C3DBaseTexture*);
};
```