# Математика

## Линейная алгебра

### 1. Матрицы

#### 1.1 Переход между системами координат





- вектор при W = 0, точка при W = 1



Z

Z

А

p

X

Базисы **A** и **C** заданы построчными матрицами.

Базис **A** задан относительно базиса **C**.

Начала **A** и **C** не совпадают (*OA*(-1 0 0) )

Вектор/точка **p** задан(а) относительно базиса **A**.

Показано, что для перехода от **A** к **C** нужно умножить

**p** на записанную построчно матрицу базиса А (при условии, что

**A** задан относительно **C**); т.о. для обратного перехода - от **C** к **A**

надо использовать обратную матрицу **A**: т.е. компонентные векторы базиса **A** должны быть записаны по-столбцово, что равносильно транспонированию матрицы **A**.

Умножение **p** на построчно записанную матрицу **A** - переход от **A** к **C** - эквивалентно линейной комбинации компонентных векторов базиса **A**, построенных относительно базиса **C**, и координат **p**.

Z

X

X

C





A =



=

=



- при взятии обратной матрицы происходит инверсия вектора

перемещения, т.о. можно просто изменить знак элементов

последней строки матрицы перемещения

#### 1.2 Матричные операции

##### 1.2.1 Свойства

**Свойства операции транспонирования матриц:**

(A^T)^T=A

(AB)^T=B^T A^T

(A^{-1})^T = (A^T)^{-1}, если [обратная матрица](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0) A^{-1} существует.

 (A+B)^T=A^T+B^T 

 \text{det}\; A=\text{det}\; A^T 

**Свойства умножения матриц на число:**

*Умножение матрицы A на число*  \lambda \in \mathcal{K} заключается в построении матрицы \lambda A = ( \lambda a_{ij} ).

* 1. 1*A* = *A*;
* 2. *(λβ)A = λ(βA)*
* 3. *(λ+β)A = λA + βA*
* 4. *λ(A+B) = λA + λB*

**Свойства сложения матриц:**

* 1.[коммутативность](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BC%D1%83%D1%82%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%8C): *A+B = B+A*;
* 2.[ассоциативность](http://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%81%D0%BE%D1%86%D0%B8%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%8C): *(A+B)+C =A+(B+C)*;
* 3.сложение с нулевой матрицей: *A + Θ = A*;
* 4.существование противоположной матрицы: *A + (-A) = Θ*;

**Свойства умножения матриц:**

* 1.[ассоциативность](http://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%81%D0%BE%D1%86%D0%B8%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%8C) *(AB)C = A(BC)*;
* 2.[некоммутативность](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BC%D1%83%D1%82%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%8C) (в общем случае): *AB \neq BA*;
* 3.произведение коммутативно для умножения на единичную или обратную матрицы: *AI = IA*; *AA-1 = A-1A*
* 4.[дистрибутивность](http://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%8C): *(A+B)C = AC + BC*, *A(B+C) = AB + AC*;
* 5.ассоциативность и коммутативность относительно умножения на число: *(λA)B = λ(AB) = A(λB)*;

##### 1.2.2 Умножение матриц

**Произведение матриц** *AB* **состоит из всех возможных комбинаций скалярных произведений строк матрицы** *A* **и столбцов матрицы** *B*. Элемент матрицы *AB* с индексами *i, j* есть скалярное произведение *i*-ой строки матрицы *A* и *j*-го столбца матрицы *B*:

= =

Количество столбцов в матрице A должно совпадать с количеством строк в матрице B, иными словами, матрица A обязана быть **согласованной**с матрицей B. Если матрица A имеет размерность m1 X n1, а матрице B имеет размерность m2 X n2, то размерность их произведения A B = C есть m1 X n2 :

1. Для умножения матрицы должны быт согласованы: n1 = m2

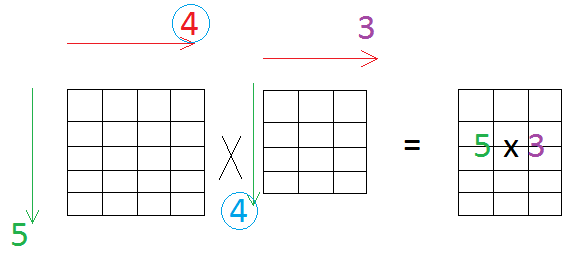
2. Размерность результирующей матрицы будет: m1 X n2



Значения на пересечениях отмеченных кружочками будут:

\begin{align}
{\color{Red}x_{1,2}} &= (a_{1,1}, a_{1,2})\cdot(b_{1,2}, b_{2,2}) \\
 &= a_{1,1}b_{1,2} + a_{1,2}b_{2,2} \\
{\color{Blue}x_{3,3}} &= (a_{3,1}, a_{3,2})\cdot(b_{1,3}, b_{2,3}) \\
 &= a_{3,1}b_{1,3} + a_{3,2}b_{2,3}
\end{align}

###### **1.2.2.1 Умножение матриц разных размерностей**



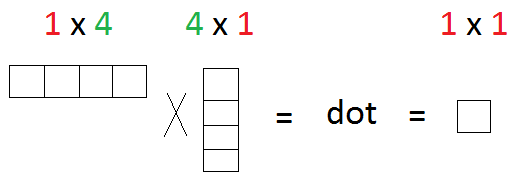
**1.2.2.1.1 Умножение векторов**

Частным случаем матрицы является вектор, следовательно векторы можно умножать используя правила умножения матриц.

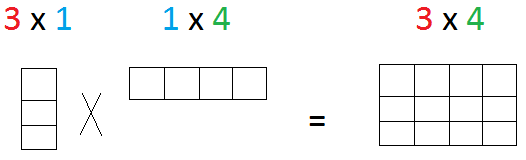
Есть два вида произведений векторов:

1) Внутреннее произведение векторов (inner product):

- тождественно скалярному произведению векторов



2) Внешенее (или тензорное) произведение векторов (outer product):



**1.2.2.1.2 Умножение вектора на матрицу**

**1.2.2.1.2.1 Умножение вектора на матрицу справа**

- для умножения вектора на матрицу справа – т.е. когда матрица находится справа – вектор должен быть представлен вектор-строкой:

**1.2.2.1.2.2 Умножение вектора на матрицу слева**

- для умножения вектора на матрицу слева – т.е. когда матрица находится слева – вектор должен быть представлен вектор-столбцом:

**1.2.2.1.2.3 Сравнение правого и левого произведений вектора и матрицы**

- как видно из формул произведения не равны друг другу

- отличие: одно произведение выступает как умножение вектора на транспонированную матрицу другого произведения; т.е. получить результат левого произведения используя правое можно умножив вектор-строку на транспонированную матрицу, также как и для получения результата правого произведения используя левое нужно транспонированную матрицу умножить на вектор-столбец

**1.2.2.1.2.4 Произведение вектора и конкатенированной матрицы**

Воспользовавшись выводами предыдущего пункта и свойствами транспонирования матриц, описанными в пункте 1.2.1, запишем формулу эквивалентности правого и левого произведений вектора и конкатенированной матрицы:

- поскольку для получения результата правого произведения используя левое нужно транспонировать матрицу, то:

- применив свойство транспонирования произведения матриц (AB)^T=B^T A^Tполучим

также эту запись подтверждает то соображение, что для равенства результата произведений порядок записи матриц-сомножителей вектор-столбца в левом произведении должен быть таким же как и в правом, но с учётом того, что матрица слева от вектор-столбца первой умножается на него, следовательно первой матрицей слева от должна быть матрица , т.к. она является первым сомножителем вектора в правом произведении, слева от нее должна быть записана матрица , т.к. она является следующим сомножителем; также эти матрицы надо транспонировать, что следует из пунктов 1.2.2.1.2.2 и 1.2.2.1.2.3

- получим:

*=*

# C++

## Наследование, виртуальные функции, полиморфизм, области видимости

{

Динамическое замещение (overriding) методов становится в полной мере актуальным в тех случ-х, когда программисту требуется задействовать полиморфное поведение классов, т.е. когда для создания объекта класса потомка используется ссылка или указатель на базовый класс. В этих случаях благодаря применению динамического замещения методов (т.е. объявлению методов с ключевым словом virtual) обращение к методу объекта ссылающегося на базовый класс, инициализированного объектом класса наследника, вызывается метод, соответствующий классу объекта, который использовался для инициализации указателя/ссылки.

В тех же случаях, когда используется статическое замещение методов поведение классов в соответствии с концепцией полиморфизма задействовано не будет. Вызываться будет метод базового класса (на который ссылается или указывает объявляемый объект), не взирая на его удалённость в древе иерархии наследников от класса родителя, в котором впервые вызываемый метод был статически объявлен.

}

Пример:

class Base

{

public:

Base(float a= 30);

~Base();

void virtual translateZ(const float a= 17);

void maketransparent();

//void virtual ShowVertices(); //if declared as virtual then ShowPolymorphicBehaviour(Base\* branch) function will call method of Successor, otherwise base class method is called

void ShowVertices(); //if declared as virtual then ShowPolymorphicBehaviour(Base\* branch) function will call method of Successor, otherwise base class method is called

protected:

void Scale(float sf=3);

void virtual Tesselate();

private:

void virtual rotateZcw(const float a= 30);

void aligntogrid();

};

class Successor: public Base

//class Successor: Base

{

public:

Successor(float a= 45);

~Successor();

void aligntogrid();// moving from private to public is possible

void virtual ShowVertices();

void virtual translateZ(const float a= 20); //overriding without explicit usage of keyword virtual

//void Tesselate();//static/dynamic (with/without keyword virtual) overriding works well - moving method from protected section to public

protected:

void virtual maketransparent();//static/dynamic (with/without keyword virtual) overriding works well - moving method from public section to protected

private:

void virtual rotateZcw(const float a= 30);

void Tesselate();//static/dynamic (with/without keyword virtual) overriding works well - moving method from protected section to private

//void virtual maketransparent();//static/dynamic (with/without keyword virtual) overriding works well - moving method from public section to private

};

class Derived2nd: public Successor

{

public:

Derived2nd();

void virtual ShowVertices();

//void ShowVertices();

};

…

main.cpp

…

void ShowPolymorphicBehaviour(Base\* branch);

void ShowPolymorphicBehaviour(Successor\* branch);

void main()

{

cout<<"Console App\n\n\n";

Base A= Base();

Successor B;

Derived2nd C;

B.translateZ();

cout<<"Console App\n\n\n";

ShowPolymorphicBehaviour(&A);

ShowPolymorphicBehaviour(&B);

ShowPolymorphicBehaviour(&C);

}

void ShowPolymorphicBehaviour(Base\* branch)

{

//branch->maketransparent();// will only call Base's method, since it is public while being protected in Successor

//(\*branch).maketransparent();// the same behaviour

branch->translateZ();

branch->ShowVertices();

}

void ShowPolymorphicBehaviour(Successor\* branch)

{

branch->translateZ();

branch->ShowVertices();

}

Сперва рассмотрим ситуацию, когда объявлена только функция:

void ShowPolymorphicBehaviour(Base\* branch);

В этом случае - учитывая, что метод void ShowVertices() в классе Base объявлен статически, в классе Successor – динамически (void virtual ShowVertices()) – для объекта С класса Derived2nd, в соответствии с принципом выбора метода базового класса при условии его статического объявления, вызов метода ShowVertices (branch->ShowVertices()) приведёт к вызову одноимённого метода базовго класса: Base::ShowVertices(); и наоборот, если в классе Base метод ShowVertices() объявлен виртуальным, то вызов branch->ShowVertices() приведёт к вызову метода объекта, которым был инициализирован фактический параметр branch: Derived2nd::ShowVertices().

Если в классе Derived2nd метод ShowVertices() не переопределяется, а наследуется от класса Successor, то в случае:

- если ShowVertices() динамически объявлен в Base: branch->ShowVertices()

вызовет Successor::ShowVertices();

- если ShowVertices() статически объявлен в Base, а в Successor –

динамически: branch->ShowVertices() вызовет Base::ShowVertices();

Если в классе Derived2nd метод ShowVertices()переопределяется (статически или виртуальным), то в случае:

- если ShowVertices() динамически объявлен в Base: branch->ShowVertices()

вызовет Derived2nd::ShowVertices();

- если ShowVertices() статически объявлен в Base, а в Successor –

динамически или статически: branch->ShowVertices() вызовет

Base::ShowVertices();

Аналогичное поведение характерно и для других вызовов функции ShowPolymorphicBehaviour():

ShowPolymorphicBehaviour(&A);

ShowPolymorphicBehaviour(&B);

Теперь рассмотрим ситуацию, когда объявлены оба варианта перегружаемой функции:

void ShowPolymorphicBehaviour(Base\* branch);

void ShowPolymorphicBehaviour(Successor\* branch);

В этом случае, для объекта A компилятором будет использован первый вариант функции void ShowPolymorphicBehaviour(Base\* branch), а для объектов В и С – второй void ShowPolymorphicBehaviour(Successor\* branch).

Анализ полиморфного поведения классов будем проводить для объекта С. Все принципы такого поведения рассмотренные ранее сохраняются и для этой ситуации.

Поскольку при подставлении в качестве факитических аргументов В и С в функцию ShowPolymorphicBehaviour(Successor\* branch) базовым классом является Successor, то в случае:

- если ShowVertices() переопределён динамически в Successor и динамически или

статически в Derived2nd и независимо от того как ShowVertices() объявлен в

Base: branch->ShowVertices() вызовет Derived2nd::ShowVertices();

- если ShowVertices() не переопределён в Successor, динамически или

статически переопределён в Derived2nd и динамически объявлен в Base:

branch->ShowVertices() вызовет Derived2nd::ShowVertices();

- если ShowVertices() не переопределён в Successor, динамически или

статически переопределён в Derived2nd и статически объявлен в Base:

branch->ShowVertices() вызовет Base::ShowVertices();

- если ShowVertices()статически переопределён в Successor, динамически или

статически переопределён в Derived2nd и динамически объявлен в Base:

branch->ShowVertices() вызовет Derived2nd::ShowVertices();

- если ShowVertices()динамически переопределён в Successor, динамически или

статически переопределён в Derived2nd и статически объявлен в Base:

branch->ShowVertices() вызовет Derived2nd::ShowVertices();

- если ShowVertices()статически переопределён в Successor, динамически или

статически переопределён в Derived2nd и статически объявлен в Base:

branch->ShowVertices() вызовет Successor::ShowVertices();

- если ShowVertices() не переопределён в Derived2nd, динамически или

статически переопределён в Successor и независимо от того как он объявлен в

Base: branch->ShowVertices() вызовет Successor::ShowVertices();

Вывод: при задействовании полиморфного поведения классов при вызове того или иного метода выбор класса, соответствующая реализация метода для которого будет использована, производится на основании атрибутов объявления вызываемого метода в базовом классе (не обязательно в классе первого объявления (это может быть просто частным случаем, если нет промежуточных классов потомков), а в классе на который ссылается/указывает ссылка/указатель объекта при её/его объявлении). Метод базового класса будет вызван, если в этом классе он объявлен статически, а метод класса наследника будет вызван, если изначальное его объявление в базовом классе было динамическим.

## Классы

- An instance of a class type is commonly called a *class object*. The individual class members are also called objects. The set of all member objects comprises a class object.

- класс и структура есть одно и тоже, т.е. используются для объявления объектов; синтаксические различия: в структуре как область видимости членов по-умолчанию - public так и спецификатор доступа при наследовании по-умолчанию - public, а в классе – во всех этих случаях по-умолчанию действует спецификатор private

- открытая часть класса называется интерфейсом

- инкапсуляция - сокрытие информации о внутренней структуре

:: - scope resolution operator – оператор разрешения области видимости

- допустимо перемещение членов класса (методов и данных) между любыми

областями видимости (private, protected, public);

- функции-друзья, объявляемые с ключевым словом friend, имеют доступ ко всем

секциям класса

- перегружаются внутри класса, как правило, операторы присваивания и унарные

операторы; бинарные операторы перегружаются функциями вне класса, но

находящимися в одной области видимости

Области видимости:

- private – закрытые члены; доступны только самому классу – в методах других классов (в т.ч. и производных) или в глобальных функциях недоступны; не наследуются, т.е. в методах производных классов недоступны

- protected – как и private доступны только методам класса, но наследуются производными классами, т.е. из методов производных классов к protected-членам базовго класса можно обращаться

### Полиморфизм и абстрактные классы

- полиморфизм – один интерфейс – множество реализаций

- указатель базового класса ссылается на объект производного класса, размещённый в куче или в глобальной (статической) памяти:

CBase \* obj = new Derived;

или

Derived obj\_d;

CBase \* obj = & obj\_d;

- п. может быть реализован только для указателей или ссылок на объекты

- п. требуется там где класс используемого объекта не известен на этапе компиляции и будет получен только во время ввыполнения

- примером ситуации когда требуется использование полиморфизма может быть следующий случай:

- допустим есть множество объектов, классы которых восходят к одному базовому классу А

- требуется отрисовать всё множество объектов

- тогда целесообразно будет заполнить некоторый контейнер, базовый тип элементов которого равен

А,

- если не использовать ключевое слово virtual то произойдёт статическое замещение методов, использующее раннее связывание, т.е. связывание на основании типа указателя, а не типа присваемого значения

- в случае когда метод объявлен с использованием ключевого слова virtual произойдёт динамическое замещение методов, использующее позднее связывание - определение вызываемого метода на основании типа присваемого значения, происходящее во время выполнения программы (run time)

- то обстоятельство что полиморфизм классов требует позднего связывания накладывает дополнительные расходы вычислительных ресурсов

- базовый класс содержащий виртуальный метод и производный класс называются полиморфными классами

- все классы родственные полиморфному классу являются полиморфными, какой бы ни была вложенность иерархии, при этом не обязательно полиморфные методы каждый раз объявлять с ключевым словом virtual

- When you override an overloaded base class member function, you hide all of the other overloaded versions of the base class member function—meaning that the only way to access the other versions of the member function is to explicitly call the base class member function (using scope resolution operator). So if you override an overloaded member function, it’s a good idea to override every version of the overloaded function.

--------

- абстрактный класс служит интерфейсом для последующих реализаций

- а.к. является полиморфным классом

- для того чтобы класс стал абстрактным в нём должен присутствовать хотя бы один чисто виртуальный (pure virtual) метод – метод, объявленный как виртуальный и в конце своего объявления имеющий символы = 0

virtual bool IsAbstract() = 0;

- чисто виртуальный метод не должен иметь определения, хотя и может

- а.к. может иметь в своём составе также и методы, имеющие реализацию

- а.к. не может иметь экземпляров, т.е. нельзя написать так: AbstrClass c; или AbstrClass \* c = new AbstrClass;

- однако можно использовать указатели на абстрактные классы для полиморфного доступа к методам производных классов

- при этом память под указатели на абстрактные классы можно выделять динамически (используя оператор new и производный класс)

- всякий класс наследующий от абстракного класса также является абстрактным

- класс производный от абстрактного должен переопределить все чисто виртуальные методы базового класса, в случае чего он перестанет быть абстрактным и сможет иметь экземпляры (заново в интерфейсе класса нужно написать прототип переопределяемого метода и затем написать его реализацию)

-----------------------------

- Virtual functions produce polymorphic behavior through references as well as through pointers

- A good rule of thumb is that if you have any virtual member functions in a class, you should make the destructor virtual, too:

Defining Virtual Destructors

When you use a pointer to a base class to point to an object of a derived class,

you have a potential problem. When you delete the pointer, only the base class’

destructor will be called for the object. This could lead to disastrous results

because the derived class’ destructor might need to free memory (as the

destructor for Boss does). The solution, as you might have guessed, is to

make the base class’ destructor virtual. That way, the derived class’ destructor

is called, which (as always) leads to the calling the base class’ destructor, giving

every class the chance to clean up after itself.

I put this theory into action when I declare Enemy’s destructor virtual.

virtual ~Enemy();

Introducing Polymorphism 351

In main(), when I delete the pointer pointing to the Boss object with the

following line, the Bossobject’s destructor is called, which frees the memory on

the heap that m\_pDamageMultiplierpoints to and displays the messageIn Boss

destructor, deleting m\_pMultiplier.

delete pBadGuy;

Then,Enemy’s destructor is called, which frees the memory on the heap that

m\_pDamagepoints to and displays the messageIn Enemy destructor, deleting

m\_pDamage.The object is destroyed, and all memory associated with the object is

freed.

-----------------------------

см. проект Inheritance and Polymorphism, M.Dawson - pp.347-356 Introducing Polymorphism - Using Abstract Classes

## Динамическое связывание - внутреннее устройство

Связывание - замена имени функции на её адрес в памяти во время компоновки - статическое связывание или во время выполнения программы - динамическое связывание (при наличии виртуальных функций).

- по умолчанию использ-ся статическое связывание (т.е. если нет слова virtual)

- если есть virtual, то связывание - динамическое - при этом на этапе компиляции не известно адрес какой ф-ии (из базовго или из производного класса) использовать для конкретного вызова, это выяснится после создания объекта конкретного класса

- если в классе есть виртуальная ф-я, то с классом связывается таблица виртуальных функций (создаётся компилятором во время компиляции) и при создании объекта первые 4 байта (как правило, но зависит от реализации) отводятся под указатель на ТВФ

- устрой-во таблиц ВФ: в одном столбце - название ф-ии, в др - адрес

- напр. есть класс DList, в к-м определены две функции length() - не виртуальная и add() - виртуальная, тогда для объекта этого класса:

при вызове length(): на стадии линковки в тех местах где компил-р встреч-т вызовы length() подставляет адрес length()

при вызове add(): в месте вызова этой ф-ии компил-р ставит набор инструкций: пойди в ТВФ объекта (пойди в его первые 4 байта, возьми от туда адрес ТВФ, в ТВФ найди строку с именем ф-ии add(), возьми адрес из др. столбца и сделай вызов туда), найди add() и вызови её - т.о. связывание будет произведено в момент выполнения программы

## Типы данных

### Базовые типы данных C++

|  |  |  |
| --- | --- | --- |
| Тип | Размер, байт | Значения |
| char | 1 | 28=256 значений символов |
| bool | 1 | true или false |
| unsigned short | 2 | 0…216-1=65535 |
| short | 2 | -32768..32767 |
| unsigned int | 4 | 0…232-1= 4294967295 |
| int | 4 | -(231)=-2147483648…231-1=2147483647 |
| unsigned long | 4 | 0…232-1= 4294967295 |
| long | 4 | -2147483648..2147483647 |
| float | 4 | 1.2e-38..3.4e38 |
| double | 8 | 2.2e-308..1.8e308 |

### Базовые типы данных Delphi

|  |  |  |
| --- | --- | --- |
| Тип | Размер, байт | Значения |
| Char | 1 | 28=256 значений символов |
| Boolean | 1 | true или false |
| Byte | 1 | 0…28-1=255 |
| ShortInt | 1 | -(27)=-128…27-1=127 |
| SmallInt | 2 | -32768..32767 |
| Word | 2 | 0…216-1=65535 |
| Integer | 4 | -(231)=-2147483648…231-1=2147483647 |
| LongInt | 4 | -2147483648..2147483647 |
| LongWord | 4 | 0…232-1= 4294967295 |
| Int64 | 8 | -(263)=-2147483648…263-1=2147483647 |
| Cardinal | 4 | 0…232-1= 4294967295 |
| Real | 8 | 5\*10-324…1.7\*10308 |
| Single | 4 | 1.5\*10-45…3.4\*1038 |
| Double | 8 | 5\*10-324…1.7\*10308 |
| Extended | 10 | 3.4\*10-4951…1.1\*104932 |
| Comp | 8 | -263..263-1 |
| Currency | 8 | ±922337203685477,5807 |

## Указатели, ссылки, массивы, динамическая и стековая память

### Динамическая и стековая память

- Стековая память, Стек – память, которая функционирует по принципу LIFO и автоматически выделяется программе для временного хранения локальных переменных, параметров функций, адресов возврата из функций. По завершении функции стековая память освобождается, т.е. значения всех локальных переменных исчезают

- Динамическая память, Свободная память (free store) – область памяти, выделямая для хранения данных, по требованию программиста

- Динам-я память, в отличие от стека, не освобождается после выхода из функций

- Контроль д. п. – её выделение и освобождение - полностью возложен на программиста

- В отличие от стековой памяти ячейкам своб-й памяти нельзя присвоить имя, доступ к ним осуществляется через указатели.

- Выделение д.п. осуществляется при помощи оператора new.

- Освобождение д.п. осуществляется при помощи оператора delete.

- Дополнительные функции C++ для работы с динамической памятью: void\*malloc(size\_t size), void\* calloc(size\_t nmemd, size\_t size), void\* realloc(void\* ptr, size\_t size), void free(void\* ptr);

### Выравнивание памяти (memory alignment)

- поскольку процессор считывает данные из ОП словами – блоками размером от 1 до 8 байт – то наиболее эффективным расположением данных в памяти будет такое, при котором адрес начала размещения данных будет кратен номинальному выравниванию их типа: данные типа int будут выделены на 4-байтной адресной границе, short – на 2-байтной и т.д.

- каждый базовый ТД имеет своё номинальное выравнивание (default alignment), которое совпадает с его размером:

float (alignement) : 4 bytes

float (size) : 4 bytes

double (alignement) : 8 bytes

double (size) : 8 bytes

double (alignement) : 8 bytes

double (size) : 8 bytes

long double (alignement) : 8 bytes

long double (size) : 8 bytes

char (alignement) : 1 byte

char (size) : 1 byte

bool (alignement) : 1 byte

bool (size) : 1 byte

int (alignement) : 4 bytes

int (size) : 4 bytes

unsigned int (alignement) : 4 bytes

unsigned int (size) : 4 bytes

short (alignement) : 2 bytes

short (size) : 2 bytes

long (alignement) : 4 bytes

long (size) : 4 bytes

unsigned long (alignement) : 4 bytes

unsigned long (size) : 4 bytes

\_\_int64 (alignement) : 8 bytes == long long

\_\_int64 (size) : 8 bytes == long long

- величина машинного слова определяется шириной (разрядностью) шины данных и/или разрядностью процессора, т.е. тем количеством информации, которое можно передать за один такт процессора

- например, для 32-битных процессоров, ширина обрабатываемого слова равна 4 байтам

- за один такт процессор может считать одно машинное слово из ОП

- т.о. на 32-битной машине данные типа int будут считаны за один такт, тогда как данные типа double потребуют двух тактов для считывания и будут выделены на 8-байтной адресной границе

- на 64-битной машине данные типа double будут считаны за один такт и размещены на 8-байтной адресной границе

- пользовательские ТД имеют размер и выравнивание, зависящие от порядка (на последних моделях процессоров, по всей видимости, порядок не имеет значения?) членов и выравнивания их типов

- так, выравнивание всей структуры (класса, объединения) будет равно размеру наибольшего ТД её членов, но если в объявлении структуры использовалась директива \_\_declspec(align(n\_байт)), то её выравнивание будет равно n\_байт (если это значение больше чем знач. выравнивания наиб-го члена стр-ры):

struct s\_tag1

{

int I;

char c;

float f;

bool b;

long l;

double d;

};

\_\_alignof(s\_tag1) == 8, т.к. наибольший размер типа данных имеет член d, т.е. sizeof(double) == 8

- размер (sizeof) *пользовательского* ТД равен сумме смещения (offset) последнего члена и его размера, а также - в случае необходимости – величины дополнения (padding), т.о., чтобы полученное в результате значение было кратно (multiple of) либо величине выравнивания наибольшего типа данных членов пользовательской структуры, либо кратно величине выравнивания, указанного для всей структуры посредством директивы \_\_declspec(align(n\_байт)), в зависимости от того, какое выравнивание больше (т.к. \_\_declspec(align(n\_байт)) может только повысить величину выравнивания)

The sizeof value for any structure is the offset of the final member, plus that member's size, rounded up to the nearest multiple of the largest member alignment value or the whole structure alignment value, whichever is greater

- дополнение данными(data padding) – дополнение памяти, выделяемой для члена пользовательского ТД, несколькими байтами, количество которых должно быть минимально достаточным для того, чтобы в сумме с размером типа данных соответствующего члена получилось значение, кратное выравниванию (размеру) типа данных следующего члена:

(sizeof(m\_i) + padding\_i) mod sizeof(m\_i+1) == 0

при этом будет обезпечено выделение каждого члена на адресной границе, соответствующей выравниванию его типа данных:

struct str1

{

int i1; 4

char c1; 1

int i2; 4

char c2; 1

double d1; 8

float f1; 4

short s1; 2

double d2; 8

short s2; 2

};

адрес\_начала (если считать от нуля) = адрес\_конца\_prev + padding

адрес\_конца = адрес\_начала + sizeof(T)

имя pad адр. начала адр. конца

i1 0 0 = 0+0 4 = 0+4

c1 0 4 = 4+0 5 = 4+1

i2 3 8 = 5+3 12 = 8+4

c2 0 12 = 12+0 13 = 12+1

d1 3 16 = 13+3 24 = 16+8

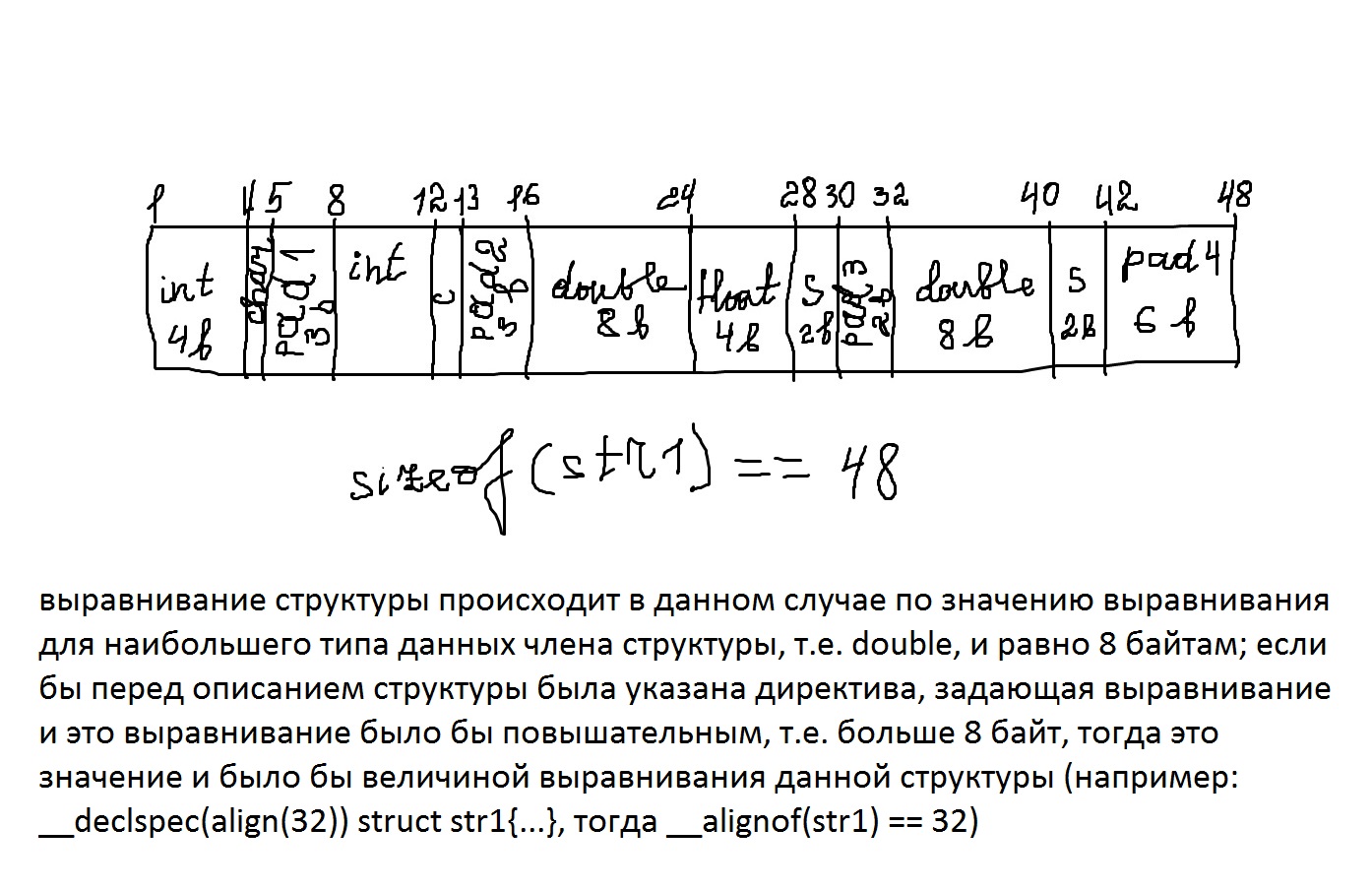
f1 0 24 = 24+0 28 = 24+4

s1 0 28 = 28+0 30 = 28+2

d2 2 32 = 30+2 40 = 32+8

s2 0 40 = 40+0 42 = 40+2

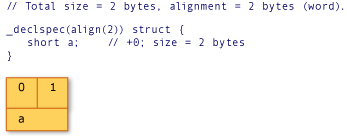
sizeof(str1) == 48 == 42+padding == 42+6 - так, чтобы размер структуры был кратным значению наибольшего ТД в структуре и при этом был ближайшим к её чистому размеру, т.е. к 42



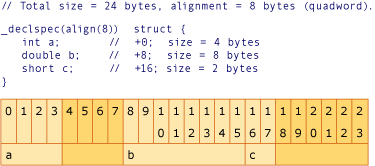
**Ещё примеры – из MSDN:**

The following four examples each declare an aligned structure or union, and the corresponding figures illustrate the layout of that structure or union in memory. Each column in a figure represents a byte of memory, and the number in the column indicates the displacement of that byte. The name in the second row of each figure corresponds to the name of a variable in the declaration. The shaded columns indicate padding that is required to achieve the specified alignment.

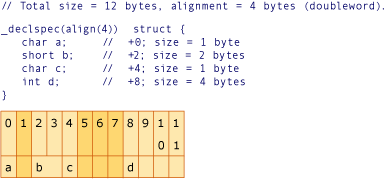
Example 1



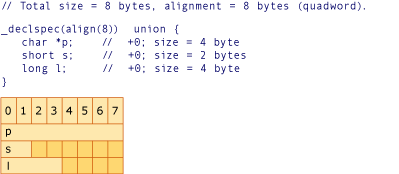
Example 2



Example 3



Example 4



### Указатели

- Указатель – это переменная, хранящая адрес ячейки памяти, в которой могут находиться данные. Например какая-либо другая переменная может хранить в этой ячейке свое значение. Т.е. указатель можно связать с какой-либо переменной, в т.ч. и с объектом.

- Путём разыменования указателя можно получить доступ к данным, хранящимся по адресу, который содержит указатель. Т.о. указатели используются для непрямого (косвенного) доступа к связанным с ними данным (переменным, объектам) и их изменения. Манипулируя указателем мы вносим изменения в объект, на который ссылается (указывает) указатель.

- Основные области применения указателей: а) размещение данных в свободных областях памяти (в динамич. пам.) и доступ к ним; б) доступ к переменным и методам классов;

в) передача параметров в функции по указателю; г) возвращение результатов функций по указателю.

- Указатели используют для передачи параметров в функции и возвращения результатов функций, что является более эффективным по сравнению с передачей параметров по значению и возвращением результата по значению.

- Особенно это эффективно при работе с объектами больших размеров.

- Передача параметров в функцию и возвращение результата функции по указателю позволяют избежать затратного копирования данных, что происходит при передаче параметров по значению и при возвращении значения из функции.

- Передавая в функции параметры по указателю, в стек копируется адрес, хранящийся в указателе, а не само значение указателя, т.о. в значительной степени повышается эффективность расходования машинных ресурсов, в отличие от передачи параметров по значению, когда в стек копируется значение фактического параметра. То же самое происходит и при возвращении результата из функции.

- Всегда после объявления указатель должен быть инициализирован в том же выражении каким-либо значением, это может быть адрес переменной, либо ноль.

- Указатель, инициализированный нулём называется пустым или нулевым указателем.

- указатели бывают типизированные и нетипизированные.

- При объявлении указатель может связываться с каким-либо типом данных, такие у-ли наз-ся типизированными. Типизированный у-ль может хранить только адрес значения указанного типа.

- Указатель связанный с типом void называется нетипизированным указателем, например void\* ptr. Такой у-ль может хранить адрес значения любого типа, однако его нельзя разыменовать, для обращения к его значению необходимо привести его к указателю на некоторый тип.

- Указатель **не** инициализированный с помощью оператора new адресует данные размещённые в стековой памяти.

### Операторы new и delete

- You can define your own versions of **new** and **delete** for a class by overloading them

- You can declare the **new** and **delete** operators with additional parameters

- When **new** and **delete** operate on class objects, the class member operator functions **new** and **delete** are called, if they have been declared

- The C++ operators for allocating and deallocating arrays of class objects are **operator new[ ]()** and**operator delete[ ]()**

- If you create a class object with the **new** operator, one of the operator functions **operator new()** or**operator new[]()** (if they have been declared) is called to create the object

- An **operator new()** or **operator new[]()** for a class is always a static class member, even if it is not declared with the keyword **static**

- It has a return type **void\*** and its first parameter must be the size of the object type and have type **std::size\_t**

**-** It cannot be virtual

- Type **std::size\_t** is an **implementation-dependent** unsigned integral type defined in the standard library header **<cstddef>**

- When you overload the **new** operator as a member function, you must declare it returning **void\***, with its first parameter of type **std::size\_t**

- You can declare additional parameters in the declaration of **operator new()** or **operator new[]()**

- Use the placement syntax to specify values for these parameters in an allocation expression

#### Простой memory pool

The following example overloads two operator **new** functions:

* X::operator new(size\_t sz): This overloads the default **new** operator by allocating memory with the C function malloc(), and throwing a string (instead of std::bad\_alloc) if malloc() fails.
* X::operator new(size\_t sz, int location): This function takes an additional integer parameter, location. This function implements a very simplistic "memory manager" that manages the storage of up to three X objects.

Static array X::buffer holds three Node objects. Each Node object contains a pointer to an X object named data and a boolean variable named filled. Each X object stores an integer called number.

When you use this **new** operator, you pass the argument location which indicates the array location of buffer where you want to "create" your new X object. If the array location is not "filled" (the data member of filled is equal to false at that array location), the **new** operator returns a pointer pointing to the X object located at buffer[location].

#include <new>

#include <iostream>

using namespace std;

class X;

struct Node {

X\* data;

bool filled;

Node() : filled(false) { }

};

class X {

static Node buffer[];

public:

int number;

enum { size = 3};

void\* operator new(size\_t sz) throw (const char\*) {

void\* p = malloc(sz);

if (sz == 0) throw "Error: malloc() failed";

cout << "X::operator new(size\_t)" << endl;

return p;

}

void \*operator new(size\_t sz, int location) throw (const char\*) {

cout << "X::operator new(size\_t, " << location << ")" << endl;

void\* p = 0;

if (location < 0 || location >= size || buffer[location].filled == true) {

throw "Error: buffer location occupied";

}

else {

p = malloc(sizeof(X));

if (p == 0) throw "Error: Creating X object failed";

buffer[location].filled = true;

buffer[location].data = (X\*) p;

}

return p;

}

static void printbuffer() {

for (int i = 0; i < size; i++) {

cout << buffer[i].data->number << endl;

}

}

};

Node X::buffer[size];

int main() {

try {

X\* ptr1 = new X;

X\* ptr2 = new(0) X;

X\* ptr3 = new(1) X;

X\* ptr4 = new(2) X;

ptr2->number = 10000;

ptr3->number = 10001;

ptr4->number = 10002;

X::printbuffer();

X\* ptr5 = new(0) X;

}

catch (const char\* message) {

cout << message << endl;

}

}

The following is the output of the above example:

X::operator new(size\_t)

X::operator new(size\_t, 0)

X::operator new(size\_t, 1)

X::operator new(size\_t, 2)

10000

10001

10002

X::operator new(size\_t, 0)

Error: buffer location occupied

The statement X\* ptr1 = new X calls X::operator new(sizeof(X)). The statement X\* ptr2 = new(0) X calls X::operator new(sizeof(X),0).

- The **delete** operator destroys an object created by the **new** operator

- If you destroy a class object with the **delete** operator, the operator function **operator delete()** or **operator delete[]()** (if they have been declared) is called to destroy the object

- The operand of **delete** must be a pointer returned by **new**

- If **delete** is called for an object with a destructor, the destructor is invoked before the object is deallocated (идёт речь о виртуальном деструкторе при полиморфизме, когда такой деструктор может быть не создан в базовом и не переопределён в производном классе; но если виртуальный деструктор переопределён, то он отработает при вызывое delete: If the delete expression is used to deallocate a class object whose static type has a virtual destructor, the deallocation function is resolved through the virtual destructor of the dynamic type of the object)

#### Полиморфизм и оператор delete

You cannot declare the **delete** operator as virtual. However you can add polymorphic behavior to your**delete** operators by declaring the destructor of a base class as virtual. The following example demonstrates this:

#include <iostream>

using namespace std;

struct A {

virtual ~A() { cout << "~A()" << endl; };

void operator delete(void\* p) {

cout << "A::operator delete" << endl;

free(p);

}

};

struct B : A {

void operator delete(void\* p) {

cout << "B::operator delete" << endl;

free(p);

}

};

int main() {

A\* ap = new B;

delete ap;

}

The following is the output of the above example:

~A()

B::operator delete

The statement delete ap uses the **delete** operator from class B instead of class A because the destructor of A has been declared as virtual.

Although you can get polymorphic behavior from the **delete** operator, the **delete** operator that is statically visible must still be accessible even though another **delete** operator might be called. For example, in the above example, the function A::operator delete(void\*) must be accessible even though the example calls B::operator delete(void\*) instead.

Virtual destructors do not have any affect on deallocation operators for arrays (**operator delete[]()**). The following example demonstrates this:

#include <iostream>

using namespace std;

struct A {

virtual ~A() { cout << "~A()" << endl; }

void operator delete[](void\* p, size\_t) {

cout << "A::operator delete[]" << endl;

::delete [] p;

}

};

struct B : A {

void operator delete[](void\* p, size\_t) {

cout << "B::operator delete[]" << endl;

::delete [] p;

}

};

int main() {

A\* bp = new B[3];

delete[] bp;

};

The behavior of the statement delete[] bp is undefined.

When you overload the **delete** operator, you must declare it as class member, returning type **void**, with the first parameter having type **void\***, as described above. You can add a second parameter of type**size\_t** to the declaration. You can only have one **operator delete()** or **operator delete[]()** for a single class.

- После удаления объекта оператором **delete**, адрес указателя на объект не меняется, но содержимое, хранящееся по этому адресу уничтожается – заполняется произвольными значениями

- Т.о. если не сделать указатель после удаления нулевым, то он станет блуждающим

- The result of trying to access a deleted object is undefined because the value of the object can change after deletion

- An **operator delete()** or **operator delete[]()** for a class is always a static member, even if it is not declared with the keyword **static**

- Its first parameter must have type **void\***

**-** Because **operator delete()** and **operator delete[]()** have a return type **void**, they cannot return a value

- The following example shows the declaration and use of the operator functions **operator new()** and**operator delete()**:

#include <cstdlib>

#include <iostream>

using namespace std;

class X {

public:

void\* operator new(size\_t sz) throw (const char\*) {

void\* p = malloc(sz);

if (p == 0) throw "malloc() failed";

return p;

}

// single argument

void operator delete(void\* p) {

cout << "X::operator delete(void\*)" << endl;

free(p);

}

};

class Y {

int filler[100];

public:

// two arguments

void operator delete(void\* p, size\_t sz) throw (const char\*) {

cout << "Freeing " << sz << " byte(s)" << endl;

free(p);

};

};

int main() {

X\* ptr = new X;

// call X::operator delete(void\*)

delete ptr;

Y\* yptr = new Y;

// call Y::operator delete(void\*, size\_t)

// with size of Y as second argument

delete yptr;

}

The above example will generate output similar to the following:

X::operator delete(void\*)

Freeing 400 byte(s)

The statement delete ptr calls X::operator delete(void\*). The statement delete yptr calls Y::operator delete(void\*, size\_t).

#### Итого:

- If **new** and **delete** are called for a class object that does not declare the operator functions **new** and**delete**, or they are called for a nonclass object, the global operators **new** and **delete** are used

- The global operators **new** and **delete** are provided in the C++ library

**-** You may define your own new operator or allocation function as a class member function or a global namespace function with the following restrictions:

* The first parameter must be of type **std::size\_t**. It cannot have a default parameter.
* The return type must be of type **void\***.
* Your allocation function may be a template function. Neither the first parameter nor the return type may depend on a template parameter.
* If you declare your allocation function with the empty exception specification throw(), your allocation function must return a null pointer if your function fails. Otherwise, your function must throw an exception of type **std::bad\_alloc** or a class derived from **std::bad\_alloc** if your function fails.

You may define your own delete operator or deallocation function as a class member function or a global namespace function with the following restrictions:

* The first parameter must be of type **void\***.
* The return type must be of type **void**.
* Your deallocation function may be a template function. Neither the first parameter nor the return type may depend on a template parameter.

The following example defines replacement functions for global namespace **new** and **delete**:

#include <cstdio>

#include <cstdlib>

using namespace std;

void\* operator new(size\_t sz) {

printf("operator new with %d bytes\n", sz);

void\* p = malloc(sz);

if (p == 0) printf("Memory error\n");

return p;

}

void operator delete(void\* p) {

if (p == 0) printf ("Deleting a null pointer\n");

else {

printf("delete object\n");

free(p);

}

}

struct A {

const char\* data;

A() : data("Text String") { printf("Constructor of S\n"); }

~A() { printf("Destructor of S\n"); }

};

int main() {

A\* ap1 = new A;

delete ap1;

printf("Array of size 2:\n");

A\* ap2 = new A[2];

delete[] ap2;

}

Вывод:

operator new with 4 bytes

Constructor of S

Destructor of S

delete object

Array of size 2:

operator new with 12 bytes

Constructor of S

Constructor of S

Destructor of S

Destructor of S

#### Оператор new

- The **new** keyword allocates memory for an object or array of objects of *type-name* from the free store and returns a suitably typed, nonzero pointer to the object. If unsuccessful, **new** returns zero or throws an exception - std::bad\_alloc

- When **new** is used to allocate memory for a C++ class object, the object's constructor is called after the memory is allocated

- The allocation-expression — the expression containing the **new** operator — does three things:

1. Locates and reserves storage for the object or objects to be allocated. When this stage is complete, the correct amount of storage is allocated, but it is not yet an object.

2. Initializes the object(s). Once initialization is complete, enough information is present for the allocated storage to be an object.

3. Returns a pointer to the object(s) of a pointer type derived from new-type-name or type-name. The program uses this pointer to access the newly allocated object.

- The **new** operator invokes the function **operator new**

**-** For arrays of any type, and for objects that are not of **class**, **struct**, or **union** types, a global function, **::operator new**, is called to allocate storage

- Class-type objects can define their own **operator new** static member function on a per-class basis (что также необходимо при использовании placement new)

- When the compiler encounters the **new** operator to allocate an object of type *type*, it issues a call to *type***::operator new( sizeof(** *type* **) )**

**-** or, if no user-defined **operator new** is defined, **::operator new( sizeof(** *type* **) )**

**-** The argument to **operator new** is of type **size\_t**. This type is defined in DIRECT.H, MALLOC.H, MEMORY.H, SEARCH.H, STDDEF.H, STDIO.H, STDLIB.H, STRING.H, and TIME.H

- behavior of throwing bad\_alloc can be changed by writing a custom exception-handling routine and calling the [\_set\_new\_handler](http://msdn.microsoft.com/en-us/library/a45x8asx(v=vs.71).aspx) run-time library function with your function name as its argument (your routine must attempt to free memory and retry the allocation)

- Use the [delete](http://msdn.microsoft.com/en-us/library/h6227113(v=vs.71).aspx) operator to deallocate the memory allocated with the **new** operator

**Grammar**

*new-expression:*

[**::**] **new** [(*placement)*] *type-name[(constructor-params)]*[*new-initializer*]  
[**::**] **new** [(*placement)*] **(** *type-name* **)** [*new-initializer*]

*placement*

Provides a way of passing additional arguments if you overload **new**.

*type-name*

Specifies type to be allocated; it can be either a built-in or user-defined type. If the type specification is complicated, it can be surrounded by parentheses to force the order of binding.

*initializer*

Provides a value for the initialized object. Initializers cannot be specified for arrays. The **new** operator will create arrays of objects only if the class has a default constructor.

- The scope-resolution operator (**::**) forces use of the global **new** operator:

**::new**

- The following example allocates and then frees a two-dimensional array of characters of size *dim* by 10. When allocating a multidimensional array, all dimensions except the first must be constant expressions that evaluate to positive values; the leftmost array dimension can be any expression that evaluates to a positive value. When allocating an array using the **new** operator, the first dimension can be zero — the **new** operator returns a unique pointer

char (\*pchar)[10] = new char[dim][10];

delete [] pchar;

- The *type-name* cannot contain **const**, **volatile**, class declarations, or enumeration declarations. Therefore, the following expression is illegal:

volatile char \*vch = new volatile char[20];

- The **new** operator does not allocate reference types because they are not objects

##### Placement new

- In C++ placement new is used to construct an object at a particular memory location or to pass additional arguments to an allocation function

- отрабатывает конструктор (объект инициализируется), память не выделяется, происходит размещение созданного объекта по указанному адресу

- в случае отсутствия перегруженных версий отрабатывает глобальная (или классовая ?) функция void\*  operator  new(size\_t nbytes, T& obj)

- placement new is useful when building a memory pool, a garbage collector or simply when performance and exception safety are paramount

- there's no danger of allocation failure since the memory has already been allocated, and constructing an object on a pre-allocated buffer takes less time

char \*buf = new char[sizeof(string)]; // pre-allocated buffer

string \*p = new (buf) string("hi"); // placement new

string \*q = new string("hi"); // ordinary heap allocation

- It is also very important to remember to #include <memory>, otherwise you might run into some terrible headaches in some platforms that do not automatically recognize placement new

- If placement arguments are used, a declaration of **operator new()** or **operator new[]()** with these arguments must exist

- Синтаксис оператора **new**, называемый **placement new**, используется в двух аспектах, при этом в зависимости от списка аргументов будет вызываться либо глобальная, либо перегруженная в классе функция **operator new**:

**1. Для вызова пользовательской функции operator new, который требует передачи списка аргументов:**

**-** The placement new syntax can be used for passing parameters to an allocation routine rather than to a constructor, i.e. for user-defined implementations of **operator new**

- it allows extra information to be passed to **operator new**

An expression with a placement field such as

T \*TObject = new ( 0x0040 ) T;

is translated to

T \*TObject = T::operator new( sizeof( T ), 0x0040 ); if class T has member operator new,

otherwise to T \*TObject = ::operator new( sizeof( T ), 0x0040 );

- Although the preceding example shows only one argument in the placement field, there is no restriction on how many extra arguments can be passed to **operator new** this way

#include <new>

using namespace std;

class X

{

public:

void\* operator new(size\_t,int, int){ /\* ... \*/ }

};

// ...

int main ()

{

X\* ptr = new(1,2) X;

}

- The original intention of the placement field was to allow hardware-dependent objects to be allocated at user-specified addresses

- Even when **operator new** has been defined for a class type, the global operator can be used by using the form of this example:

T \*TObject =::new TObject;

**2. Для создания объекта (или массива объектов) и размещения его в уже выделенном участке памяти:**

- При этом оператор **new** не выделяет память, а получает своим аргументом адрес на уже выделенную каким-либо образом память (например, на стеке или через malloc). Происходит размещение объекта путем вызова конструктора, и объект создается в памяти по указанному адресу. Часто такой метод применяют, когда у класса нет конструктора по умолчанию и при этом нужно создать массив объектов:

class A

{

public:

A(int x){}

~A(){}

};

const int n = 50;

A\* placementMemory = static\_cast<A\*>(operator new[] (n \* sizeof(A))); // используется форма оператора new для создания массива

for (int i = 0; i < n; i++)

{

new (placementMemory + i) A(rand()); //здесь память для объекта не выделяется, но инициализируется; используется глобальная функция **operator new**

}

//!!деинициализация памяти

for (int i = 0; i < n; i++)

{

placementMemory[i].~A();

}

operator delete[] (placementMemory);

пример:

void construct(T \* const p, const T& t) const

{

   void \* const pv = static\_cast<void \*>(p);

      new (pv) T(t); // в участок памяти адресуемый указателем p помещается

// объект класса T, инициализированный значением t ;

// используется глобальная функция **operator new**

}

- The placement syntax is commonly used to invoke the global placement **new** function.

- The global placement **new** function initializes an object or objects at the location specified by the placement argument in the placement new expression

- This location must address storage that has previously been allocated by some other means, because the global placement new function does not itself allocate memory

- In the following example, no new memory is allocated by the calls new(whole) X(8); new(seg2) X(9); or new(seg3) X(10);

- Instead, the constructors X(8), X(9), and X(10) are called to reinitialize the memory allocated to the buffer whole

**-** Because placement new does not allocate memory, you should not use delete to deallocate objects created with the placement syntax (по моему не совсем верное утверждение)

**-** You can only delete the entire **memory pool** (delete whole), or you can keep **the memory buffer**, but destroy the object stored in it by explicitly calling a destructor

#include <new>

class X

{

public:

X(int n): id(n){ }

~X(){ }

private:

int id;

// ...

};

int main()

{

char\* whole = new char[ 3 \* sizeof(X) ]; // a 3-part buffer (memory pool)

X \* p1 = new(whole) X(8); // fill the front

char\* seg2 = &whole[ sizeof(X) ]; // mark second segment

X \* p2 = new(seg2) X(9); // fill second segment

char\* seg3 = &whole[ 2 \* sizeof(X) ]; // mark third segment

X \* p3 = new(seg3) X(10); // fill third segment

p2->~X(); // clear only middle segment, but keep the buffer

// ...

return 0;

}

##### Функция operator new()

- назначение функции operator new() – выделение памяти; после выделения память не инициализируется, т.е. конструктор не вызывается; затем происходит вызов operator new() с placement семантикой, который уже инициализирует переданную ему память объектом, вызывая конструктор его класса:

оператор Foo\* p = new Foo() функционально соответствует следующему коду:

// don't catch exceptions thrown by the allocator itself

void\* raw = operator new(sizeof(Foo)); // выделение памяти

// catch any exceptions thrown by the ctor

try {

p = new(raw) Foo(); // call the ctor with raw as this

}

catch (...) {

// oops, ctor threw an exception

operator delete(raw);

throw; // rethrow the ctor's exception

}

- there is no restriction on how many extra arguments can be passed to **operator new**

- When a statement such as the following is encountered in a program, it translates into a call to the function **operator new**:

char \*pch = new char[BUFFER\_SIZE];

- repeated calls to **operator new**return different pointers

- There are  two scopes for **operator new** function:

- global **::operator new**

- class *class-name***::operator new**

- The first argument to **operator new**must be of type **size\_t** (a type defined in STDDEF.H), and the return type is always **void \***

- The global **operator new**function is called when the **new** operator is used to allocate objects of built-in types, objects of class type that do not contain user-defined **operator new**functions, and arrays of any type

- When the **new** operator is used to allocate objects of a class type where an **operator new**is defined, that class's **operator new**is called

- An **operator new**function defined for a class is a static member function (which cannot, therefore, be virtual) that hides the global **operator new**function for objects of that class type

- Consider the case where **new** is used to allocate and set memory to a given value:

// spec1\_the\_operator\_new\_function1.cpp

#include <malloc.h>

#include <memory.h>

class Blanks

{

public:

Blanks(){}

void \*operator new( size\_t sztAllocateBlock, char chInit );

};

void \*Blanks::operator new( size\_t stAllocateBlock, char chInit )

{

void \*pvTemp = malloc( sztAllocateBlock );

if( pvTemp != 0 )

memset( pvTemp, chInit, sztAllocateBlock );

return pvTemp;

}

// For discrete objects of type Blanks, the global operator new function

// is hidden. Therefore, the following code allocates an object of type

// Blanks and initializes it to 0xa5

int main()

{

Blanks \*a5 = new(0xa5) Blanks;

return a5 != 0;

}

The argument supplied in parentheses to **new** is passed to Blanks::operator new as the chInit argument. However, the global **operator new**function is hidden, causing code such as the following to generate an error:

Blanks \*SomeBlanks = new Blanks;

- In Visual C++ 5.0 and earlier, nonclass types and all arrays (regardless of whether they were of **class** type) allocated using the **new** operator always used the global **operator new**function.

Beginning with Visual C++ 5.0, the compiler supports member array **new** and **delete** operators in a class declaration. For example:

// spec1\_the\_operator\_new\_function2.cpp

class MyClass

{

public:

void \* operator new[] (size\_t)

{

return 0;

}

void operator delete[] (void\*)

{

}

};

int main()

{

MyClass \*pMyClass = new MyClass[5];

delete [] pMyClass;

}

#### Оператор delete

- Memory that is dynamically allocated using the **new** operator can be freed using the **delete** operator

- The [delete operator](http://msdn.microsoft.com/en-us/library/h6227113.aspx) invokes the function [operator delete](http://msdn.microsoft.com/en-us/library/248aa748.aspx), which frees memory back to the available pool

- Using the **delete** operator also causes the class destructor (if there is one) to be called

- There are global and class-scoped **operator delete**functions

-Only one **operator delete**function can be defined for a given class; if defined, it hides the global **operator delete**function

- The global **operator delete**function is always called for arrays of any type

- The global **operator delete**function, if declared, takes a single argument of type **void \***, which contains a pointer to the object to deallocate. The return type is **void** (**operator delete**cannot return a value)

- Для объектов не классового типа ([class](http://msdn.microsoft.com/en-us/library/w5c4hyx3.aspx), [struct](http://msdn.microsoft.com/en-us/library/64973255.aspx), or [union](http://msdn.microsoft.com/en-us/library/5dxy4b7b.aspx)), вызывается глобальный оператор delete (глобальная функция delete)

- For objects of class type, the name of the deallocation function is resolved in global scope if the delete expression begins with the unary scope resolution operator (::)

- Otherwise, the delete operator invokes the destructor for an object prior to deallocating memory (if the pointer is not null).

-The delete operator can be defined on a per-class basis

- if there is no such definition for a given class, the global operator delete is invoked

- If the delete expression is used to deallocate a class object whose static type has a virtual destructor, the deallocation function is resolved through the virtual destructor of the dynamic type of the object

- The global **operator delete**function, if declared, takes a single argument of type **void \***, which contains a pointer to the object to deallocate. The return type is **void** (**operator delete**cannot return a value):

void operator delete( void \* );

void operator delete( void \*, size\_t );

Only one of the preceding two variants can be present for a given class.

The first form works as described for global **operator delete.**

The second form takes two arguments, the first of which is a pointer to the memory block to deallocate and the second of which is the number of bytes to deallocate.

The second form is particularly useful when an **operator delete**function from a base class is used to delete an object of a derived class.

- The **operator delete**function is static; therefore, it cannot be virtual

- The **operator delete** function obeys access control, as described in [Member-Access Control](http://msdn.microsoft.com/en-us/library/kktasw36.aspx)

- to detect "memory leakage", the global **new** and **delete** operators are redefined to count allocation and deallocation of memory, for example:

// spec1\_the\_operator\_delete\_function1.cpp

// compile with: /EHsc

// arguments: 3

#include <iostream>

using namespace std;

int fLogMemory = 0; // Perform logging (0=no; nonzero=yes)?

int cBlocksAllocated = 0; // Count of blocks allocated.

// User-defined operator new.

void \*operator new( size\_t stAllocateBlock ) {

static int fInOpNew = 0; // Guard flag.

if ( fLogMemory && !fInOpNew ) {

fInOpNew = 1;

clog << "Memory block " << ++cBlocksAllocated

<< " allocated for " << stAllocateBlock

<< " bytes\n";

fInOpNew = 0;

}

return malloc( stAllocateBlock );

}

// User-defined operator delete.

void operator delete( void \*pvMem ) {

static int fInOpDelete = 0; // Guard flag.

if ( fLogMemory && !fInOpDelete ) {

fInOpDelete = 1;

clog << "Memory block " << cBlocksAllocated--

<< " deallocated\n";

fInOpDelete = 0;

}

free( pvMem );

}

int main( int argc, char \*argv[] ) {

fLogMemory = 1; // Turn logging on

if( argc > 1 )

for( int i = 0; i < atoi( argv[1] ); ++i ) {

char \*pMem = new char[10];

delete[] pMem;

}

fLogMemory = 0; // Turn logging off.

return cBlocksAllocated;

}

Итого:

The **delete** operator destroys the object created with **new** by deallocating the memory associated with the object.

The **delete** operator has a **void** return type. It has the syntax:

>>-+----+--delete--*object\_pointer*------------------------------><

'-::-'

The operand of **delete** must be a pointer returned by **new**, and cannot be a pointer to constant. Deleting a null pointer has no effect.

The **delete[]** operator frees storage allocated for array objects created with **new[]**. The **delete**operator frees storage allocated for individual objects created with **new**.

It has the syntax:

>>-+----+--delete--[--]--*array*---------------------------------><

'-::-'

The result of deleting an array object with **delete** is undefined, as is deleting an individual object with **delete[]**. The array dimensions do not need to be specified with **delete[]**.

The result of any attempt to access a deleted object or array is undefined.

If a destructor has been defined for a class, **delete** invokes that destructor. Whether a destructor exists or not, **delete** frees the storage pointed to by calling the function **operator delete()** of the class if one exists.

The global **::operator delete()** is used if:

* The class has no **operator delete()**.
* The object is of a nonclass type.
* The object is deleted with the **::delete** expression.

The global **::operator delete[]()** is used if:

* The class has no **operator delete[]()**
* The object is of a nonclass type
* The object is deleted with the **::delete[]** expression.

- The default global **operator delete()** only frees storage allocated by the default global **operator new()**

- The default global **operator delete[]()** only frees storage allocated for arrays by the default global **operator new[]()**.

## Преобразование типов

<http://www.cplusplus.com/doc/tutorial/typecasting/>

### dynamic\_cast

- использ-ся только для преобр-й между ссылками и ук-ми на родственные классы

- назначение этого каста - убедиться, что рез-т преобр-я ук-т на правильный полный объект целевого типа (result of the type conversion points to a valid complete object of the destination pointer type)

- upcast (Derived->Base) и downcast (Base->Derived)

- upcast не имеет практического значения, т.к. выполняется неявно сам по себе

- downcast вернёт не nullptr только для полиморфных классов и только, если преобр-й ук-ль адресует объект производного класса:

dynamic\_cast<Derived\*>(obj\_ptr) != nullptr if typeof(\*obj\_ptr) == Derived

- в случае неудачного преобразования возвращается nullptr или исключение bad\_cast (если типами были ссылки)

- dynamic\_cast can also perform the other implicit casts allowed on pointers: casting null pointers between pointers types (even between unrelated classes), and casting any pointer of any type to a void\* pointer

- может: T\* -> void\* (делается и так, неявным образом, поэтому в dynamic\_cast здесь нет потребности)

- не может: void\* -> T\*

- можно применить для преобр-я не родственных классов, но для этого преобразуемый класс должен иметь виртуальный метод; такой код скомпилируется, но рез-м будет nullptr

### static\_cast

- делает всё то же самое, только не проверяет полиморфность классов, следовательно - быстрее

- родственность классов проверяется во время компиляции (попытка скастовать между неродственными классами приведёт к ошибке компиляции)

- не может делать преобразования между ук-ми на неродственные классы, код не скомпилируется

- может: T\* -> void\* (повторяет функциональность неявного преобразования)

- может: void\* -> T\*

Основное:

- работают только для родственных классов

- dynamic\_cast делает downcast только для полиморфных классов; проверяет в рантайме адресует ли преобразуемый у-ль базового класса объект производного класса

- dynamic\_cast возвращает значение, говорящее об успешности преобразования - не nullptr в случае успешного преобразования

- static\_cast такого значения не возвращает, т.е. преобразование не произойдёт и не будет какого-либо индикатора этого; всё что произойдёт - адреса обоих у-й станут одинковыми

- static\_cast работает быстрее, т.к. не делает проверок в рантайме, но и не даёт средств для определения успешности преобразования

- static\_cast может void\* -> T\* (преобразование обратное неявному), чего не может dynamic\_cast

- оба преобразования не могут T\* -> U\*

### reinterpret\_cast

- в отличии от 2х предыдущих может преобразовывать любые у-ли к любым

- классы не должны быть родственными

- T\* -> U\*

- reinterpret\_cast converts any pointer type to any other pointer type, even of unrelated classes

- all pointer conversions are allowed: neither the content pointed nor the pointer type itself is checked

- the operation result is a simple binary copy of the value from one pointer to the other

## Аллокаторы и пулы памяти

Аллокаторы – это объекты, предоставляющие интерфейс (в случае stateless аллокаторов) либо механизмы (в случае state аллокаторов) для хранения данных, выделения и освобождения памяти

Пулы памяти – это объекты, предоставляющие механизмы хранения данных, выделения и освобождения памяти под эти данные

### Аллокаторы

- default memory allocator is generally optimized for dealing with large, variable-sized memory blocks

- if you're allocating a big block of memory, holding on to it for a long time, and then releasing it all at once (as is the case with vectors and deques), it may be hard to beat the minimalist wrapper around *new* and*delete*. If you're using node-based containers (such as maps, sets, or lists), allocators optimized for smaller objects are probably a better way to go.

- In some situations, however, the standard allocator can be heavily inefficient in terms of both time and space

- The problem is that *std::allocator* is, by definition, a general-purpose allocator, and yet there's really no such thing as general-purpose memory usage. Different programs use memory differently. Even within the same program, different areas have different memory usage patterns

- Scott Meyers, for instance, talks about using alternatives to*std::allocator* (see *Effective STL*). Andrei Alexandrescu devotes an entire chapter of *Modern C++ Design: Generic Programming and Design Patterns Applied*(Addison-Wesley, 2001) to an allocator for small objects. The Boost library provides two wrappers around its*pool\_allocator* that are *std::allocator*s

- It would be nice to have a whole family of allocators, each optimized for a different memory usage pattern, ideally sharing as much code as possible. It should be possible to switch from one allocation strategy to another as easily as possible

-  each program has different memory usage patterns, and this makes it difficult to write a general-purpose allocator that's good for all cases

- An Estimate of the Store Size Necessary for Dynamic Storage Allocation," by J.M. Robson, *Journal of the ACM*, 18(3):416-423, 1971

- However, there are common usage patterns that are seen again and again, especially with standard containers. Once you know how your container is going to be used, you ought to be able to specify the allocation strategy that performs best with your particular usage pattern. In addition, the environment in which the code is being executed plays a part in design decisions. For instance, if an allocator is only going to be used in single-threaded environments, there's no need to worry about protecting its data against access from multiple threads

- The family of allocators just described is a perfect example of a case where policy-based design can really shine. Knowledge of how the container will be used, performance requirements on the container, the size of the objects that will be allocated, and the environment in which the code will be executed are all generally known at compile time. Ideally, users of the allocator could specify the right combination of policies, and an allocator configured to perform well for the given situation would be generated at compile time

### Пулы памяти

- пул памяти (memory pool/memory arena) – это объект, имеющий заранее выделенный буфер данных (напр. void\* или char [n\*sizeof(T)] – n и T передаются как параметры шаблона), который используется для размещения произвольного количества других объектов (рзмещение происходит при помощи placement синтаксиса оператора new)

- п.п. выделяет большой участок памяти (при помощи определённого системного обращения можно выделить участок в: shared memory, persistent memory, stack memory и т.д.), которую он потом раздаёт частями

- класс п.п. может иметь следующий вид:

class Pool {

public:

void\* alloc(size\_t nbytes);

void dealloc(void\* p);

private:

*...data members used in your pool object...*

};

void\* Pool::alloc(size\_t nbytes)

{

*...your algorithm goes here...*

}

void Pool::dealloc(void\* p)

{

*...your algorithm goes here...*

}

- теперь можно использовать данный п.п. следующим образом:

Pool pool;

*...*

void\* raw = pool.alloc(sizeof(Foo));

Foo\* p = new(raw) Foo();

или просто

Foo\* p = new(pool.alloc(sizeof(Foo))) Foo();

- задание п.п. в виде объекта класса позволяет создавать множество различных п.п. нежели чем иметь один большой пул, разделяемый всеми пользователями

- преимущества:

- если нужно часто выделять/освобождать память: однократное выделение п.п. с последующим размещением в нём объектов, а при необходимости удалить каждый объект можно просто удалить весь п.п.

- при многопоточности, можно создать пул разделяемой памяти (shared memory – специальная память, выделяемая ОС, являющаяся общей для нескольких процессов) и выдавать её по частям

- можно создать пул в стековой памяти – используется функция alloca (реализована не для всех платформ), выделяющая участок в стеке; т.о. по завершении ф-ии эта память автоматически освобождается; однако при этом не во всех случаях отработают деструкторы

- для упрощения синтаксиса использования placement new с пулом памяти нужно перегрузить глобальный оператор new следующим образом:

inline void\* operator new(size\_t nbytes, Pool& pool)

{

return pool.alloc(nbytes);

}

Now when the compiler sees new(pool) Foo(), it calls the above operator new and passes sizeof(Foo) and pool as parameters

- для уничтожения объектов Foo надо явно вызывать деструктор и затем явно освобождать память:

void sample(Pool& pool)

{

Foo\* p = new(pool) Foo();

...

p->~Foo(); // explicitly call dtor

pool.dealloc(p); // explicitly release the memory

}

Это сопряжено с 3мя проблемами:

1. произойдёт утечка памяти, если Foo::Foo() бросит исключение

2. отличие синтаксиса уничтожения объекта/освобождения памяти от привычного ведёт к

ошибкам

3. пользователь должен всегда знать какой пул соответствует какому объекту

1. Устранение потенциальной утечки памяти при удалении/освобождении:

При использовании предоставляемого компилятором оператора new, например Foo\* p = new Foo, компилятор генерирует специальный код, учитывающий возможность того, что конструктор бросит исключение. Следующий код функционально подобен коду компилятора, который будет сгенерирован при обработке оператора Foo\* p = new Foo :

Foo\* p;

// исключения при выделении памяти не учитываются (не перехватываются)

void\* raw = operator new(sizeof(Foo));

// перехватить исключения, бросаемые конструктором

try

{

p = new(raw) Foo(); // вызов конструктора с this == raw

}

catch(…)

{

operator delete(raw); // удалить raw в случае исключения, брошенного конструктором

throw; // перебросить исключение конструктора выше

}

Смысл в том, что компилятор освобождает память при бросании конструтктором исключения. Однако в том же случае, но при использовании placement семантики, компилятор не освободит память оператором delete вызовет метод Pool::deallocate() :

при обработке оператора Foo\* p = new(pool) Foo будет сгенерирован код, функционально подобный следующему:

void\* raw = operator new(sizeof(Foo), pool); // pool.alloc(sizeof(Foo));

Foo\* p = new(raw) Foo; // в случае броска исключения конструктором метод pool.dealloc(raw) вызван не будет

Освобождение памяти raw оператором delete не происходит, т.к. использовалась перегруженная функция operator new(……), значит для выделения памяти применялась нестандартная стратегия, поэтому для её освобождения также должна быть применена перегуженная функция operator delete(……). Когда компилятор встречает new(pool) Foo, то он ищет соответствующий operator delete. Найдя таковой, он помещает обращение к конструктору в защищённый блок, как это происходит и в обычном случае. Перегруженный оператор delete должен иметь следующий вид:

void operator delete(void\* p, Pool& pool)

{

pool.dealloc(p);

}

Теперь при обработке оператора Foo\* p = new(pool) Foo будет сгенерирован код, функционально подобный следующему:

Foo\* p;

void\* raw = new(sizeof(Foo), pool); // pool.alloc(sizeof(Foo));

try

{

p = new(raw) Foo;

}

catch

{

operator delete(raw, pool); // сработает после броска исключения конструктором и освободит

память raw

throw;

}

Проблемы 2 и 3 решаются одновременно. Идея состоит в том, чтобы неявно ассоциировать пул с каждым выделением памяти. При этом пул, связанный с глобальным оператором new будет равен NULL. Перегружается оператор new для фактического параметра типа Pool, где в первые n (как правило 4) байт выделяемой под объект памяти помещается указатель на пул и возвращается указатель на начало участка, в котором будет размещён сам объект после обращения к системному placement new. А в перегруженном операторе delete указатель обратно смещается на n байт и если в этих n-байтах хранится указатель на пул, то вызывается метод Pool::dealloc(), иначе память освобождается функцией free():

void\* operator new(size\_t nbytes)

{

if (nbytes == 0)

nbytes = 1; // so all alloc's get a distinct address

void\* ans = malloc(nbytes + 4); // overallocate by 4 bytes

\*(Pool\*\*)ans = NULL; // use *NULL* in the global *new*

return (char\*)ans + 4; // don't let users see the *Pool\**

}

void\* operator new(size\_t nbytes, Pool& pool)

{

if (nbytes == 0)

nbytes = 1; // so all alloc's get a distinct address

void\* ans = pool.alloc(nbytes + 4); // overallocate by 4 bytes

\*(Pool\*\*)ans = &pool; // put the *Pool\** here

return (char\*)ans + 4; // don't let users see the *Pool\**

}

void operator delete(void\* p)

{

if (p != NULL) {

p = (char\*)p - 4; // back off to the *Pool\**

Pool\* pool = \*(Pool\*\*)p;

if (pool == NULL)

free(p); // note: 4 bytes left of the original *p*

else

pool->dealloc(p); // note: 4 bytes left of the original *p*

}

}

## C++11/14

### auto

- Starting with Visual C++ 2010, the**auto** keyword declares a variable whose type is deduced from the initialization expression in its declaration:

auto declarator initializer;

- у этого ключевого слова два значения: Before Visual C++ 2010, the **auto** keyword declares a variable in the *automatic* storage class; that is, a variable that has a local lifetime; см. auto Keyword (Storage-Class Specifier)

auto declarator;

**Автоматический вывод типа:**

- Deduces the type of a declared variable from its initialization expression:

auto declarator initializer;

### decltype

- The **decltype** type specifier yields the type of a specified expression

- Use **auto** and **decltype** to declare a template function whose return type depends on the types of its template arguments

- Or, use **auto** and **decltype** to declare a template function that wraps a call to another function, and then returns the return type of the wrapped function

**Синтаксис:**

decltype( expression )

**Возвращаемое значение:**

Тип параметра expression

### Trailing return types

- позволяет использовать decltype и auto для самостоятельного вывода компилятором типа возврата функции (Use the *alternative function declaration syntax* that is shown later, the **auto** keyword, and the **decltype** type specifier to declare a *late-specified* return type. The late-specified return type is determined when the declaration is compiled, instead of when it is coded):

template<typename T, typename U>

auto myFunc(T&& t, U&& u) -> decltype (forward<T>(t) + forward<U>(u))

{ return forward<T>(t) + forward<U>(u); };

- Another benefit of using trailing return types is the improvement of readability and maintainability of programs

### Rvalue references, move semantics and perfect forwarding

#### Move semantics

Основная статья на MSDN: [Rvalue Reference Declarator: &&](http://msdn.microsoft.com/en-us/library/dd293668.aspx)

Rvalue references enable you to distinguish an lvalue from an rvalue. Lvalue references and rvalue references are syntactically and semantically similar, but they follow somewhat different rules.

- всё, адрес чего можно взять является левым значением; адрес, чего нельзя взять, является правым значением (временные объекты на стеке, литералы)

Rvalue references support the implementation of *move semantics*, which can significantly increase the performance of your applications. Move semantics enables you to write code that transfers resources (such as dynamically allocated memory) from one object to another. Move semantics works because it enables resources to be transferred from temporary objects that cannot be referenced elsewhere in the program.

To implement move semantics, you typically provide a *move constructor,*and optionally a move assignment operator (**operator=**), to your class. Copy and assignment operations whose sources are rvalues then automatically take advantage of move semantics. Unlike the default copy constructor, the compiler does not provide a default move constructor. For more information about how to write a move constructor and how to use it in your application, see [How to: Write a Move Constructor](http://msdn.microsoft.com/en-us/library/dd293665.aspx).

You can also overload ordinary functions and operators to take advantage of move semantics. Visual C++ 2010 introduces move semantics into the Standard Template Library (STL)

Move semantics also helps when the compiler cannot use Return Value Optimization (RVO) or Named Return Value Optimization (NRVO). In these cases, the compiler calls the move constructor if the type defines it. For more information about Named Return Value Optimization, see [Named Return Value Optimization in Visual C++ 2005](http://go.microsoft.com/fwlink/?LinkId=131571).

Consider the example of inserting an element into a **vector** object. If the capacity of the **vector** object is exceeded, the **vector** object must reallocate memory for its elements and then copy each element to another memory location to make room for the inserted element. When an insertion operation copies an element, it creates a new element, calls the copy constructor to copy the data from the previous element to the new element, and then destroys the previous element. Move semantics enables you to move objects directly without having to perform expensive memory allocation and copy operations.

To take advantage of move semantics in the **vector** example, you can write a move constructor to move data from one object to another.

For more information about the introduction of move semantics into the STL in Visual C++ 2010, see [Standard C++ Library Reference](http://msdn.microsoft.com/en-us/library/cscc687y.aspx).

##### **NRVO**

<http://msdn.microsoft.com/en-us/library/ms364057.aspx>

Typically, when a method returns an instance of an object, a temporary object is created and copied to the target object via the copy constructor. The C++ standard allows the elision of the copy constructor (even if this results in different program behavior), which has a side effect of enabling the compiler to treat both objects as one (see section 12.8. Copying class objects, paragraph 15; see[Reference](http://msdn.microsoft.com/ru-ru/library/ms364057.aspx#nrvo_cpp05_topic4ref)). The Visual C++ 8.0 compiler makes use of the flexibility that the standard provides and adds a new feature: Named Return Value Optimization (NRVO). NRVO eliminates the copy constructor and destructor of a stack-based return value. This optimizes out the redundant copy constructor and destructor calls and thus improves overall performance. It is to be noted that this could lead to different behavior between optimized and non-optimized programs (see the [Optimization Side Effects](http://msdn.microsoft.com/ru-ru/library/ms364057.aspx#nrvo_cpp05_topic4) section).

There are some cases in which the optimization will not take place (see the [Optimization Limitations](http://msdn.microsoft.com/ru-ru/library/ms364057.aspx#nrvo_cpp05_topic3) section for samples). The more common ones are:

* Different paths returning different named objects.
* Multiple return paths (even if the same named object is returned on all paths) with EH states introduced.
* The named object returned is referenced in an inline asm block.

##### **Применение std::move к параметру, который является правозначной ссылкой, внутри функции**

**- значение переданное в параметре является правознаной ссылкой, но сам параметр является левым значением**

**The compiler treats a named rvalue reference as an lvalue and an unnamed rvalue reference as an rvalue**

When you write a function that takes an rvalue reference as its parameter, that parameter is treated as an lvalue in the body of the function. The compiler treats a named rvalue reference as an lvalue because a named object can be referenced by several parts of a program; it would be dangerous to allow multiple parts of a program to modify or remove resources from that object. For example, if multiple parts of a program try to transfer resources from the same object, only the first part will successfully transfer the resource.

Внутри ф-ии её параметр, хотя имеет тип правозначой ссылки, посклольку у него есть имя, а следовательно, можно взять его адрес, сам является левым значением.

По этой причине мы применяем std::move внутри функции к её аргументам, т.к. хотим сохранить их правозначность, поскольку мы знаем, что переданное в параметре значение было правым (к правозначным ссылкам привязываются только правые значения).

#### Пример написания конструктора и оператора присвоения, оператора сложения и функции, использующих семантику перемещения

Основная статья: [How to: Write a Move Constructor](http://msdn.microsoft.com/en-us/library/dd293665.aspx)

#include <iostream>

#include <typeinfo.h>

using namespace std;

class VisualEffect

{

public:

VisualEffect(double amplitude) : representation(amplitude) {}

VisualEffect(VisualEffect&& vfx)

{

cout << "VisualEffect(VisualEffect&& vfx) MOVE CTOR   ----- 1\n\n";

\*this = std::move(vfx);

cout << "VisualEffect(VisualEffect&& vfx) MOVE CTOR   ----- 2\n\n";

}

VisualEffect& operator=(VisualEffect&& vfx)

{

cout << "VisualEffect& operator=(VisualEffect&& vfx) ----- 1\n\n";

if (&vfx != this)

{

// Free any existing pointed resources

// ...

// Make a deep copy of vfx

// ...

// Assign the data members of the source object vfx to default values. This prevents the destructor from freeing resources (such as memory) multiple times

// ...

}

cout << "VisualEffect& operator=(VisualEffect&& vfx) ----- 2\n\n";

return \*this;

}

VisualEffect operator+(VisualEffect&& vfx)

{

cout << "VisualEffect operator+(VisualEffect&& vfx)   ----- 1\n\n";

VisualEffect tmp(std::move(vfx));

// Add vfx representation to this representation

// ...

cout << "VisualEffect operator+(VisualEffect&& vfx)   ----- 2\n\n";

return \*this; // will use NRVO (but could use move ctor to return value if there were multiple return paths with one return name or multiple return paths with different return names)

}

private:

double representation;

};

void modifyFX(VisualEffect&& vfx, VisualEffect& vfxOut)

{

// doing operations on vfx, i.e. changing it

// ...

vfxOut = std::move(vfx); // making vfx an rvalue, since we know that expression passed in vfx is rvalue (only rvalue can bind to such param type),

 // but identifier vfx itself is an lvalue; and passing it using move semantics to move assign operator

}

void main()

{

VisualEffect compoundVFX(0);

modifyFX(VisualEffect(1) + VisualEffect(9), compoundVFX);

}

Вывод:

VisualEffect operator+(VisualEffect&& vfx) ----- 1

VisualEffect(VisualEffect&& vfx) MOVE CTOR ----- 1

VisualEffect& operator=(VisualEffect&& vfx) ----- 1

VisualEffect& operator=(VisualEffect&& vfx) ----- 2

VisualEffect(VisualEffect&& vfx) MOVE CTOR ----- 2

VisualEffect operator+(VisualEffect&& vfx) ----- 2

VisualEffect& operator=(VisualEffect&& vfx) ----- 1

VisualEffect& operator=(VisualEffect&& vfx) ----- 2

#### Perfect forwarding (идеальное перенаправление)

Perfect forwarding reduces the need for overloaded functions and helps avoid the forwarding problem.

struct W

{

W(int&, int&) {}

};

struct X

{

X(const int&, int&) {}

};

struct Y

{

Y(int&, const int&) {}

};

struct Z

{

Z(const int&, const int&) {}

};

Suppose you want to write a generic function that generates objects.  Ordinarily, to solve this problem, you must create an overloaded version of the factory function for every combination of A& and const A& parameters. Rvalue references enable you to write one version of the factory function, as shown in the following example:

template <typename T, typename A1, typename A2>

T\* factory(A1&& a1, A2&& a2)

{

return new T(std::forward<A1>(a1), std::forward<A2>(a2));

}

int main()

{

int a = 4, b = 5;

W\* pw = factory<W>(a, b);

X\* px = factory<X>(2, b);

Y\* py = factory<Y>(a, 2);

Z\* pz = factory<Z>(2, 2);

delete pw;

delete px;

delete py;

delete pz;

}

Правозначные ссылки в данном контексте, когда присутствует вывод типа, и тип записан как T (без дополнительных примесей – модификаторов и т.д.), называются универсальными (в терминологии Скотта Мейерса) или (в терминах стандарта C++ 14) перенаправляющими (forwarding references). Они встречаются везде , где есть вывод типа и запись типа можно абстрактно обозначить как T: auto/typedef/decltype/templates.

##### **Универсальные ссылки**

- Universal reference (URef): T&& and type deduction (template/auto/decltype) for T – can take rvalues, lvalues and everything else: consts, volatiles, etc. – therefore type of URef param can be deduced (during compile time) as being an LRef (T&) or RRef (T&&): if actual param was an lvalue URef is substituted for LRef (T&), and if actual param was an rvalue URef is replaced by RRef (T&&)

- Rvalue reference (RRef): if there’s T&& and no type deduction – can take only rvalues

- URefs can be assigned with either lvalues or with rvalues

- URefs are much more powerful than RRefs which is obvious from their defenition; URefs can take anything you pass to them, therefore you should have either URef function or several overloads – one for only rvalues (RRef), set of overloads for lvalues (const, without const) and all of these overloads will grow exponentially with increasing of the number of parameters

- But you shouldn’t (although you can) have URef version simultaneously with RRef and LRef versions, because URef version usually covers all your needs

- URefs essentially are forwarding references, i.e. you should use std::forward on URef param inside the function:

template<class T>

void doWork(T&& arg)

{

ops and exprs using std::forward<T>(arg)

}

- std::forward retains param’s initial type: if arg was an lvalue then after applying st::forward it’ll have LRef type and hence it’ll only be able to be copied, but if arg was an rvalue then std::forward turns it back into an rvalue so move semantics can be used on it

- Inside the function or method designed only for rvalues, i.e. it takes a named RRef param of concrete type(concrete\_T&& param\_name – i.e. no type deduction for T), that param is treated as lvalue, ‘cause it has a name and therefore we can take its address (and all lvalues allow to take their address). But we know that a function with such a signature can only be passed an rvalue, therefore inside this function we must turn param (which has become an lvalue due to its having a name) back to an rvalue as it was at the moment of passing into the function. This is done by using std::move on that param:

void doWork(Widget&& arg)

{

ops and exprs using std::move(arg)

}

- Therefore move semantics is now applyable to arg as it was initially planned for.

- The only time you don’t want to use std::move on a param is when you don’t wanna it to be moved but to be copied instead

- But there are cases when having all formally needed attributes of URef doesn’t mean it is a URef. The subtelty is in mandatory condition for something to be a URef – the condition of type obscurity, i.e. the type mustn’t be known before the call to the function. In template classes such as std::vector there are usually two overloads of methods, one taking an LRef and another taking RRef param:

template<class T, class Allocator=allocator<T>>

class vector

{

public:

…

void push\_back(const T& x); // LRef (copy lvalues)

void pus\_back(T&& x); // RRef (move rvalues)

template<class… Args>

void emplace\_back(Args&&… args); // URef (forward everything)

};

Regardless of the fact that RRef version of push\_back method has template type T and double ref sign which implies it being a universal reference, it is not, because type deduction isn’t taking place there, since by the moment the push\_back is called the factual type of T will have already been inferred, so push\_back will be called with T replaced by a concrete type for wich vector container was instantiated for. So given this context we get push\_back method that can take only rvalues.

Hence there are two overloads of push\_back, one for lvalues and one for rvalues. This is needed for implementing of copy and move semantics respectively of element addition operation to a vector.

This principle is applied to free functions either. The only purpose of overloading a function on RRef parameter is to implement move semantics and thus inside the function that param must be wrapped with std::move()

Method emplace\_back has its own template type Args…, which will only be known when method is called, therefore Args&& is a URef (&& token and type deduction) and param args must be forwarded inside the function with std::forward.

Here in std::vector template we simply do not have the possibility to use a URef push\_back and that’s why we have two overloads of it

- With URefs use std::forward<Type\_to\_deduce>(URef\_param) and with RRefs use std::move(RRef\_param)

###### **Универсальные ссылки и перегрузка – перегружайте только если требуется специальная обработка const LRef или const RRef (const T& param или const T&& param)**

Если написать перегрузку (шаблонной) функции, принимающей универсальную ссылку, изменив тип параметра на const T&, то такая функция будет вызываться только для константных левых или правых значений:

#include <iostream>

#include <typeinfo.h>

using namespace std;

template<class T> void f(const T& param)

{

cout << "const T&: T is "<< typeid(T).name() <<"\n\n";

}

template<class T> void f(T&& param)

{

cout << "T&& \n\n";

}

void main()

{

const int&& v = 111;

f(v); // результат: const T&: T is int

}

Однако, это нужно только когда есть потребность в особом обращении с константными левыми или правыми значениями.

Иначе же, достаточно наличия одной шаблонной ф-ии, принимающей универсальную ссылку, т.к. унив-е ссылки могут принимать значения любого типа, включая const T&. Любые модификаторы типа аргумента, такие как const или volatile будут сохранены в процессе вывода типа T.

###### **Как универсальные ссылки становятся левозначными и правозначными ссылками**

Формула этой трансформации проста:

универсальная\_ссылка = правозначная\_ссылка + правила\_вывода\_типа\_для\_универсальных\_ссылок + сворачивание\_ссылок

(URef = RRef + type deduction for URefs + reference collapsing)

**Правила вывода типа для универсальных ссылок**

Когда правозначные ссылки появляются в универсальном контексте, т.е. когда мы имеем T&&, в силу вступают особые правила вывода типа T, отличающиеся от правил вывода типа при инстансинге шаблона:

1) Если переданное в параметре значение было левым, то тип T будет определён как T& (вместо T будет подставлен тип значения):

param was lvalue: T => T&

2) Если переданное в параметре значение было правым, то тип T будет оставлен без изменений:

param was rvalue: T => T

**Сворачивание ссылок (reference collapsing)**

Взятие ссылки от ссылки является запрещённой операцией. Такая ситуация вызовет ошибку стадии компиляции, если программист явным образом возьмёт ссылку от ссылки. Однако неявно это может делать сам компилятор в тех случаях, когда происходит выведение типа. Ситуацию спасают следующие правила преобразования ссылок:

1) && & => &

2) & && => &

3) & & => &

4) && && => &&

Или, если обобщить, то получим всего 2 правила:

1) Всё, что содержит левозначную ссылку даёт её же.

2) Правозначная ссылка с правозначной ссылкой дают правозначную ссылку.

Т.о. в примере выше:

void process(Widget& lvp); // обработка левых значений

void process(Widget&& rvp); // обработка правых значений

template<class T> void redirectTOappropriateProcess(T&& p)

{

process(std::forward<T>(p));

}

Widget w;

redirectTOappropriateProcess(w);

redirectTOappropriateProcess(std::move(w));

мы получаем следующую последовательность этапов вывода типа T:

1) применение правил вывода типа для универсальных ссылок:

а) для redirectTOappropriateProcess(w):

w является lvalue => T выводится как Widget&

б) для redirectTOappropriateProcess(std::move(w)):

std::move(w) является rvalue => T выводится как Widget

2) применение сворачивания ссылок:

а) для redirectTOappropriateProcess(w):

Widget& && => Widget&

б) для redirectTOappropriateProcess(std::move(w)):

Widget && => Widget&&

#### Обобщение

В зависимости от контекста применения правозначные ссылки могут быть связаны:

а) только с правыми значениями rvalue

б) как с правыми так и с левыми значениями lvalue (тип которых может включать любые модификаторы: const, volatile и др.)

В случае (a) мы имеем следующий вид объявления правозначной ссылки (rvalue refefrence - RRef):

конкретный\_тип&& имя ,

где конкретный\_тип – невыводимый, известный на момент объявления тип данных.

Например:

Matrix&& m44;

Правозначные ссылки нужны для перемещения значений вместо копирования, что даёт значительный прирост производительности. Их используют в связке с безусловным оператором приведения типа std::move, результатом которого, независимо от типа переданного аргумента, всегда будет приведение аргумента к RRef:

std::move – unconditional cast to an rvalue

пример концептуальной реализации на C++ 11:

template<class T> typename remove\_reference<T>::type&&

move(T&& param)

{

using ReturnType = typename remove\_reference<T>::type&&;

return static\_cast<ReturnType>(param);

}

пример концептуальной реализации на C++ 14:

template<class T>

decltype(auto) move(T&& param)

{

using ReturnType = typename remove\_reference<T> &&;

return static\_cast<ReturnType>(param);

}

Как видно из приведенного кода, тип параметра очищается от возможного наличия ссылки ( отбрасывается возможно присутствующий амперсанд), затем из «чистого» типа делают правозначную ссылку (добавляются два амперсанда) и кастуют к нему аргумент, т.е. делают его правозначной ссылкой.

В случае (б) мы имеем дело с правозначными ссылками, которые можно назвать универсальными ссылками (URef), т.к. в данном контексте правозначные ссылки могут быть связаны с любым значением. Универсальный контекст применения правозначных ссылок требует обязательного наличия следующих двух признаков:

1) тип данных ссылки должен быть выводимым, следовательно это могут быть следующие 4 контекста: шаблоны, auto, typedef и decltype

2) запись типа в объявлении шаблона/auto/typedef/decltype должна иметь только следующий вид:

T&&

какое-либо отличие в записи типа ведёт к невыполнению второго требования (например: const T&& или vector<T&&>) и мы получим уже правозначную ссылку на конкретный тип данных.

Универсальные ссылки служат для реализации концепции идеального перенаправления (perfect forwarding): способа сохранения правозначности (rvalueness) передаваемого значения. URef станет RRef (конкретный\_тип&&), если будет инициализирована правым значением (rvalue) и станет LRef (конкретный\_тип&, т.е. lvalue передаются на стек по ссылке как и обычно), если будет инициализирована левым значением (lvalue). То, чем станет URef зависит от исходного типа выражения, передаваемого параметру с типом URef. Определить каким – левым или правым – значением является переданное выражение можно по выводимому типу T.

Для возможности осуществить идеальное перенаправление с помощью URefs используют условный оператор приведения типа std::forward, который приводит аргумент к RRef, если его тип является RRef, иначе он возвращает аргумент как есть, без изменения его типа, т.е. возвращает lvalue reference – LRef:

std::forward – conditional cast to an rvalue

пример концептуальной реализации на C++ 11:

template<class T> T&& forward(T&& param)

{

if (is\_lvalue\_reference<T>::value) return param;

else return move(param);

}

пример написания функции, использующей идеальное перенаправление:

void process(Widget& lvp); // обработка левых значений

void process(Widget&& rvp); // обработка правых значений

template<class T> void redirectTOappropriateProcess(T&& p)

{

process(std::forward<T>(p));

}

Widget w;

redirectTOappropriateProcess(w); // вызов process для левых значений - первой перегрузки

redirectTOappropriateProcess(std::move(w)); // вызов process для правых значений - второй

перегрузки

Т.о. идеальное перенаправление позволяет нам написать всего лишь один интерфейс, который будет подходить для всех случаев.

#### Обратная совместимость с C++ 98

Типы данных, написанные на C++ 98 не содержат конструкторов и операторов присвоения с семантикой перемещения, для которой используются правозначные ссылки. Параметрами для указанных методов являются левозначные ссылки на константу:

const type& param

Т.к. правые значения (правозначные ссылки) совместимы с левозначными ссылками, то приведение объекта при помощи std::move к правому значению перед, например, присвоением или передачей в качестве аргумента, в случаях наличия перегруженных методов, не всегда гарантирует, что этот объект будет перемещён. Попросту, в результате разрешения перегрузки (overload resolution) может быть выбран метод с параметром левой ссылкой на константу:

class string

{

public:

…

string& operator=(const string& rhs); // copy assignment

string& operator=(string&& rhs); // move assignment

…

};

string s1;

const string s2;

s1 = std::move(s2); // make s2’s type const string&&

Строку s2 привели к првозначной ссылке и хотят присвоить другой строке. Однако константная правозначная ссылка не может быть связана с неконстантной правозначной ссылкой (это было бы нарушением константной корректности), но она совместима с константной левозначной ссылкой (работают правила сворачивания ссылок), т.к. уже говорилось – правые значения (правозначные ссылки) совместимы с левозначными ссылками.

Следовательно, будет вызван оператор присвоения копировщик. Т.о., там где ожидалось перемещение объекта, произошло его копирование, хотя использовался оператор move.

Данное свойство совместимости правозначных ссылок с левозначными позволяет добиться совместимости нового кода со старым. Просто перемещение объектов, не имеющих перегруженных методов для семантики перемещения, заменяется их копированием.

- objects that don’t know how to be moved decay into copy operations for backward compatibility

Однако, эта же обратная совместимость, может послужить причиной того, что ваш объект, который поддерживает перемещение будет скопирован, если он был объявлен константным. И это вполне обоснованно, т.к. перемещение изменяет объект.

Поэтому объекты, которые будут перемещаться, не должны быть константными.

- you shouldn’t declare objects const if you don’t want to move from them, because move changes objects

#### Использование оператора noexcept в прототипах функций с семантикой перемещения

Видео: An Effective C++11,14 Sampler (Scott Meyers), 26:24

noexcept specifier (which is an operator, taking and returning bool) - use it when you can gaurantee strong exception safety for given function - allows compiler to produce more efficient code than w/o it.

До внедрения семантики перемещения использовалось копирование, так например, добавление нового элемента в контейнер, не имеющий свободного места для этого, приводило к выделению нового участка памяти и последующему копированию туда существующих элементов. Поскольку использовалось копирование, то при возникновении исключительной ситуации (во время выделения памяти под новое хранилище или копирования) исходные данные гарантированно оставались неповреждёнными, что позволяло давать сильную гарантию (strong exception safety guaranty). С приходом семантики перемещения исходные данные перемещаются и в случае возникновения исключительной ситуации они могут быть утеряны, т.о. гарантия становится потенциально базовой (basic safety guaranty).

Ранее для указания сильной гарантии использовался спецификатор пустого исключения throw(), который говроил, что данная функция не вызывает исключительных ситуаций. При этом если всё же внутри вызываемой ф-ии возникало исключение, то:

1) её стек передавался вызывающей ф-ии,

2) затем отрабатывал метод std::unexpected, который вызывал std::terminate.

Теперь, для передачи той же семантики нужно использовать оператор noexcept, наличие которого обуславливало следующее поведение программы в случае возникновения исключительной ситуации:

1) возможно стек бросившей ф-ии передавался вызывавшей ф-ии,

2) отрабатывал метод std::unexpected, который вызывал std::terminate.

В случае с noexcept возможность не передачи стека выше позволяет компилятору генерировать более эффективный код. Т.о. везде, где возможно гарантровать, что ф-ия не бросит исключение, следует использовать оператор noexcept.

В C++ 11 версии STL добавление элементов в контейнер имеет сильную гарантию, если операция move для базового типа контейнера имеет семантику пустого исключения (noexcept). Внутри, например, vector<T>::push\_back выбор между перемещением и копированием в зависимости от exception семантики оператора std::move для базового типа T, происходит благодаря использованию метода std::move\_if\_noexcept

- noexcept является оператором, т.к. сам может принимать значения типа bool и возвращать результат типа bool; т.о. можно записывать условные выражения с использованием самого же оператора noexcept, что позволяет задавать exception safety данной ф-ии в зависимости от exception safety других ф-ий

### std::back\_insert\_iterator <iterator>

<http://en.cppreference.com/w/cpp/iterator/back_insert_iterator>

template< class Container >

class back\_insert\_iterator : public [std::iterator](http://en.cppreference.com/w/cpp/iterator/iterator)< [std::output\_iterator\_tag](http://en.cppreference.com/w/cpp/iterator/iterator_tags),

                                                   void, void, void, void >

std::back\_insert\_iterator is an [OutputIterator](http://en.cppreference.com/w/cpp/concept/OutputIterator) that appends to a container for which it was constructed. The container's push\_back() member function is called whenever the iterator (whether dereferenced or not) is assigned to. Incrementing the std::back\_insert\_iterator is a no-op.

- каждое присвоение итератору (разименованному или нет) вызывает push\_back у контейнера

**Member types**

|  |  |
| --- | --- |
| **Member type** | **Definition** |
| container\_type | Container |

**Member functions**

|  |  |
| --- | --- |
| [(constructor)](http://en.cppreference.com/w/cpp/iterator/back_insert_iterator/back_insert_iterator) | constructs a new back\_insert\_iterator  (public member function) |
| [**operator=**](http://en.cppreference.com/w/cpp/iterator/back_insert_iterator/operator%3D) | inserts an object into the associated container  (public member function) |
| [**operator\***](http://en.cppreference.com/w/cpp/iterator/back_insert_iterator/operator*) | no-op  (public member function) |
| [**operator++operator++**(int)](http://en.cppreference.com/w/cpp/iterator/back_insert_iterator/operator%2B%2B) | no-op  (public member function) |

**Member objects**

|  |  |
| --- | --- |
| **Member name** | **Definition** |
| container (protected) | a pointer of type Container\* |

Inherited from [std::iterator](http://en.cppreference.com/w/cpp/iterator/iterator)

**Member types**

|  |  |
| --- | --- |
| **Member type** | **Definition** |
| value\_type | void |
| difference\_type | void |
| pointer | void |
| reference | void |
| iterator\_category | [std::output\_iterator\_tag](http://en.cppreference.com/w/cpp/iterator/iterator_tags) |

Пример:

#include <iostream>

#include <iterator>

#include <algorithm>

#include <cstdlib>

#include <vector>

int main()

{

std::vector<int> v;

std::generate\_n(std::back\_insert\_iterator<std::vector<int>>(v), // can be simplified

10, [](){ return std::rand() % 10; });        // with std::back\_inserter

for (int n : v)

std::cout << n << ' ';

std::cout << '\n';

}

### std::back\_inserter <iterator>

<http://en.cppreference.com/w/cpp/iterator/back_inserter>

template< class Container >  
[std::back\_insert\_iterator](http://en.cppreference.com/w/cpp/iterator/back_insert_iterator)<Container> back\_inserter( Container& c );

- back\_inserter is a convenience function template that constructs a [std::back\_insert\_iterator](http://en.cppreference.com/w/cpp/iterator/back_insert_iterator) for the containerc with the type deduced from the type of the argument

|  |  |  |
| --- | --- | --- |
| **c** | - | container that supports a push\_back operation |

returns a [std::back\_insert\_iterator](http://en.cppreference.com/w/cpp/iterator/back_insert_iterator) which can be used to add elements to the end of the container c

Пример:

- последние три элемента массива заполняются значением -1

#include <iostream>

#include <vector>

#include <algorithm>

#include <iterator>

int main()

{

std::vector<int> v{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

std::fill\_n(std::back\_inserter(v), 3, -1);

for (int n : v)

std::cout << n << ' ';

}

Вывод:

1 2 3 4 5 6 7 8 9 10 -1 -1 -1

### std::insert\_iterator <iterator>

<http://en.cppreference.com/w/cpp/iterator/insert_iterator>

template< class Container >

class insert\_iterator : public [std::iterator](http://en.cppreference.com/w/cpp/iterator/iterator)< [std::output\_iterator\_tag](http://en.cppreference.com/w/cpp/iterator/iterator_tags),

                                              void,void,void,void >

std::insert\_iterator is an [OutputIterator](http://en.cppreference.com/w/cpp/concept/OutputIterator) that inserts elements into a container for which it was constructed, at the position pointed to by the supplied iterator. The container's insert() member function is called whenever the iterator (whether dereferenced or not) is assigned to. Incrementing the std::insert\_iterator is a no-op.

- каждое присвоение итератору (разыменованному или нет) вызывает insert() контейнера и вставляет эелемент в позицию, указываемую итератором

Ctor:

explicit insert\_iterator( Container& c, typename Container::iterator i );

Initializes the underlying pointer to the container to std::addressof(c) and the interlying iterator to iter.

**Parameters**

|  |  |  |
| --- | --- | --- |
| **c** | - | container to initialize the inserter with |
| **iter** | - | iterator to initialize the inserter with |

**Member types**

|  |  |
| --- | --- |
| **Member type** | **Definition** |
| container\_type | Container |

**Member functions**

|  |  |
| --- | --- |
| [(constructor)](http://en.cppreference.com/w/cpp/iterator/insert_iterator/insert_iterator) | constructs a new insert\_iterator  (public member function) |
| [**operator=**](http://en.cppreference.com/w/cpp/iterator/insert_iterator/operator%3D) | inserts an object into the associated container  (public member function) |
| [**operator\***](http://en.cppreference.com/w/cpp/iterator/insert_iterator/operator*) | no-op  (public member function) |
| [**operator++operator++**(int)](http://en.cppreference.com/w/cpp/iterator/insert_iterator/operator%2B%2B) | no-op  (public member function) |

**Member objects**

|  |  |
| --- | --- |
| **Member name** | **Definition** |
| container (protected) | a pointer of type Container\* |
| iter (protected) | an iterator of type Container::iterator |

Inherited from [std::iterator](http://en.cppreference.com/w/cpp/iterator/iterator)

**Member types**

|  |  |
| --- | --- |
| **Member type** | **Definition** |
| value\_type | void |
| difference\_type | void |
| pointer | void |
| reference | void |
| iterator\_category | [std::output\_iterator\_tag](http://en.cppreference.com/w/cpp/iterator/iterator_tags) |

Пример:

- копирование всех элементов вектора и вставка в список начиная со второго элемента (insert\_iterator указывает на второй элемент списка)

#include <vector>

#include <list>

#include <iostream>

#include <iterator>

#include <algorithm>

int main()

{

    std::vector<int> v{1,2,3,4,5};

    std::list<int> l{-1,-2,-3};

    std::copy(v.begin(), v.end(), // may be simplified with std::inserter

        std::insert\_iterator<std::list<int>>(l, std::next(l.begin())));

// или упрощённая версия

//std::copy(v.begin(), v.end(), std::inserter(l, std::next(l.begin())));

    for (int n : l)

        std::cout << n << ' ';

    std::cout << '\n';

}

Вывод:

-1 1 2 3 4 5 -2 -3

### std::inserter <iterator>

<http://en.cppreference.com/w/cpp/iterator/inserter>

template< class Container >  
[std::insert\_iterator](http://en.cppreference.com/w/cpp/iterator/insert_iterator)<Container> inserter( Container& c, typename Container::iterator i );

inserter is a convenience function template that constructs a [std::insert\_iterator](http://en.cppreference.com/w/cpp/iterator/insert_iterator) for the container c and its iterator i with the type deduced from the type of the argument.

- используют для упрощённого получения std::insert\_iterator, тип которого выводится по переданному контейнеру c

**Parameters**

|  |  |  |
| --- | --- | --- |
| **c** | - | container that supports a insert operation |
| **i** | - | iterator in c indicating the insertion position |

**Return value**

A [std::insert\_iterator](http://en.cppreference.com/w/cpp/iterator/insert_iterator) which can be used to insert elements into the container c at the position indicated by i.

Пример:

- в список начиная со второго элемента вставляется три раза -1; в мультимножество начиная с первого элемента три раза вставляется -1

#include <iostream>

#include <list>

#include <algorithm>

#include <iterator>

#include <set>

int main()

{

    std::list<int> l{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    std::multiset<int> s;

    std::fill\_n(std::inserter(l, std::next(l.begin())), 3, -1);

    std::fill\_n(std::inserter(s, s.begin()), 3, -1);

    for (int n : l) {

        std::cout << n << ' ';

    }

    std::cout << '\n';

    for (int n : s) {

        std::cout << n << ' ';

    }

    std::cout << '\n';

}

Вывод:

1 -1 -1 -1 2 3 4 5 6 7 8 9 10

-1 -1 -1

### std::front\_insert\_iterator <iterator>

<http://en.cppreference.com/w/cpp/iterator/front_insert_iterator>

template< class Container >

class front\_insert\_iterator : public [std::iterator](http://en.cppreference.com/w/cpp/iterator/iterator)< [std::output\_iterator\_tag](http://en.cppreference.com/w/cpp/iterator/iterator_tags),

                                                   void,void,void,void >

std::front\_insert\_iterator is an [OutputIterator](http://en.cppreference.com/w/cpp/concept/OutputIterator) that prepends elements to a container for which it was constructed. The container's push\_front() member function is called whenever the iterator (whether dereferenced or not) is assigned to. Incrementing the std::front\_insert\_iterator is a no-op.

- присвоение итератору (разименованному или нет) вызывает push\_front у контейнера

**Member types**

|  |  |
| --- | --- |
| **Member type** | **Definition** |
| container\_type | Container |

**Member functions**

|  |  |
| --- | --- |
| [(constructor)](http://en.cppreference.com/w/cpp/iterator/front_insert_iterator/front_insert_iterator) | constructs a new front\_insert\_iterator  (public member function) |
| [**operator=**](http://en.cppreference.com/w/cpp/iterator/front_insert_iterator/operator%3D) | inserts an object into the associated container  (public member function) |
| [**operator\***](http://en.cppreference.com/w/cpp/iterator/front_insert_iterator/operator*) | no-op  (public member function) |
| [**operator++operator++**(int)](http://en.cppreference.com/w/cpp/iterator/front_insert_iterator/operator%2B%2B) | no-op  (public member function) |

**Member objects**

|  |  |
| --- | --- |
| **Member name** | **Definition** |
| container (protected) | a pointer of type Container\* |

Inherited from [std::iterator](http://en.cppreference.com/w/cpp/iterator/iterator)

**Member types**

|  |  |
| --- | --- |
| **Member type** | **Definition** |
| value\_type | void |
| difference\_type | void |
| pointer | void |
| reference | void |
| iterator\_category | [std::output\_iterator\_tag](http://en.cppreference.com/w/cpp/iterator/iterator_tags) |

Пример:

#include <vector>

#include <deque>

#include <iostream>

#include <iterator>

#include <algorithm>

int main()

{

std::vector<int> v{ 1, 2, 3, 4, 5 };

std::deque<int> d;

std::copy(v.begin(), v.end(),

std::front\_insert\_iterator<std::deque<int>>(d)); // or std::front\_inserter(d)

// или упрощённый вариант

//std::copy(v.begin(), v.end(), std::front\_inserter(d));

for (int n : d)

std::cout << n << ' ';

std::cout << '\n';

}

Вывод:

5 4 3 2 1

### std::front\_inserter <iterator>

<http://en.cppreference.com/w/cpp/iterator/front_inserter>

template< class Container >  
[std::front\_insert\_iterator](http://en.cppreference.com/w/cpp/iterator/front_insert_iterator)<Container> front\_inserter( Container& c );

front\_inserter is a convenience function template that constructs a [std::front\_insert\_iterator](http://en.cppreference.com/w/cpp/iterator/front_insert_iterator) for the container c with the type deduced from the type of the argument.

- используется как средство упрощённого получения std::fron\_insert\_iterator, т.к. требует всего лишь объект контейнера, по которому сам выводит его тип

**Parameters**

|  |  |  |
| --- | --- | --- |
| **c** | - | container that supports a push\_front operation |

**Return value**

A [std::front\_insert\_iterator](http://en.cppreference.com/w/cpp/iterator/front_insert_iterator) which can be used to add elements to the beginning of the container c

Пример:

- вставка в начало вектора значения -1 три раза

#include <iostream>

#include <deque>

#include <algorithm>

#include <iterator>

int main()

{

std::deque<int> v{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

std::fill\_n(std::front\_inserter(v), 3, -1);

for (int n : v)

std::cout << n << ' ';

}

Вывод:

-1 -1 -1 1 2 3 4 5 6 7 8 9 10

### std::prev std::next <iterator>

<http://en.cppreference.com/w/cpp/iterator/prev>

<http://en.cppreference.com/w/cpp/iterator/next>

template< class BidirIt >

BidirIt prev( BidirIt it,

              typename [std::iterator\_traits](http://en.cppreference.com/w/cpp/iterator/iterator_traits)<BidirIt>::difference\_type n = 1 );

template< class ForwardIt >

ForwardIt next( ForwardIt it,

                typename [std::iterator\_traits](http://en.cppreference.com/w/cpp/iterator/iterator_traits)<ForwardIt>::difference\_type n = 1 );

Return the nth predecessor/successor of iterator it.

- используется своместно с auto для вывода возвращаемого типа

### std::advance <iterator>

<http://en.cppreference.com/w/cpp/iterator/advance>

template< class InputIt, class Distance >  
void advance( InputIt& it, Distance n );

Increments given iterator it by n elements.

If n is negative, the iterator is decremented. In this case, InputIt must meet the requirements of[BidirectionalIterator](http://en.cppreference.com/w/cpp/concept/BidirectionalIterator), otherwise the behavior is undefined.

### std::distance <iterator>

<http://en.cppreference.com/w/cpp/iterator/distance>

template< class InputIt >

typename [std::iterator\_traits](http://en.cppreference.com/w/cpp/iterator/iterator_traits)<InputIt>::difference\_type

    distance( InputIt first, InputIt last );

Returns the number of elements between first and last.

The behavior is undefined if last is not reachable from first by (possibly repeatedly) incrementing first.

- используется своместно с auto для вывода возвращаемого типа

Пример:

#include <iostream>

#include <iterator>

#include <vector>

int main()

{

std::vector<int> v{ 3, 1, 4 };

auto distance = std::distance(v.begin(), v.end());

std::cout << distance << '\n';

}

### std::(c)begin std::(c)end <iterator>

<http://en.cppreference.com/w/cpp/iterator/begin>

<http://en.cppreference.com/w/cpp/iterator/end>

|  |  |  |
| --- | --- | --- |
| template< class C >  auto begin( C& c ) -> decltype(c.begin()); | (1) | (since C++11) |
| template< class C >  auto begin( const C& c ) -> decltype(c.begin()); | (1) | (since C++11) |
|  | (2) |  |
| template< class T, size\_t N >  T\* begin( T (&array)[N] ); | (since C++11)  (until C++14) |
| template< class T, size\_t N >  constexpr T\* begin( T (&array)[N] ); | (since C++14) |
| template< class C >  constexpr auto cbegin( const C& c ) -> decltype(std::begin(c)); | (3) | (since C++14) |

Возвращает (константный) итератор на начало/конец контейнера c.

- используется своместно с auto для вывода возвращаемого типа

- in addition to being included in <iterator>, std::begin/std::end are guaranteed to become available if any of the following headers are included: <array>, <deque>, <forward\_list>, <list>, <map>, <regex>, <set>, <string>,<unordered\_map>, <unordered\_set>, and <vector>.

- можно создать пользовательские перегрузки данных шаблонов

Пример:

#include <iostream>

#include <vector>

#include <iterator>

int main()

{

std::vector<int> v = { 3, 1, 4 };

auto vi = std::begin(v);

std::cout << \*vi << '\n';

int a[] = { -5, 10, 15 };

auto ai = std::begin(a);

std::cout << \*ai << '\n';

}

### std::tuple

Основные источники: B. Stroustrup C++ 11 FAQ <http://www.stroustrup.com/C++11FAQ.html>, MSDN: tuple class <http://msdn.microsoft.com/en-us/library/bb982837.aspx>

- по сути, является вариантным статическим массивом, т.е. может содержать разнотипные элементы, заполнение которыми происходит на стадии компиляции (во время выполнения добавлять новые элементы нельзя)

Пример явного определения кортежа:

**tuple<string,int> t2("Kylling",123);**

Основные вспомогательные методы:

1) The element types of a tuple can be explicitly specified or be deduced using **make\_tuple()**:

**auto t = make\_tuple(string("Herring"),10, 1.23);** // t will be of type

// tuple<string,int,double>

2) the elements can be accessed by (zero-based) index using **get()**:

**string s = get<0>(t);**

**int x = get<1>(t);**

**double d = get<2>(t);**

- a **pair** can be used to initialize a **tuple**, but the opposite isn't the case

- the comparison operators (==, !=, <, <=, >, and >=) are defined for tuples of comparable element types

### std::for\_each <algorithm>

Основные статьи: MSDN for\_each <http://msdn.microsoft.com/en-us/library/e5sk9w9k.aspx> ,

cppreference std::for\_each <http://en.cppreference.com/w/cpp/algorithm/for_each>

- принимает два итератора, задающих диапазон итерируемых значений коллекции, и функиоеальный объект - функтор (всё что является функциональным объектом: пользовательский функтор, т.е. экземпляр класса с перегруженным оператором вызова ф-ии; лямбда-функцию или рез-т оболочек std::mem\_fn и std::bind), который будет применён к каждому значению итерируемой коллекции

template< class InputIt, class UnaryFunction >  
UnaryFunction for\_each( InputIt first, InputIt last, UnaryFunction f );

- Applies the given function object f to the result of dereferencing every iterator in the range [first, last), in order.

If InputIt is a mutable iterator, f may modify the elements of the range through the dereferenced iterator. If f returns a result, the result is ignored.

template<class InputIterator, class Function>

Function for\_each(

InputIterator \_First,

InputIterator \_Last,

Function \_Func

);

- returns \_Func, as if calling std::move(\_Func)

Пример 1:

- два пользовательских функтора; возврат рез-та из for\_each с использованием перегрузки оператора приведения типа – возврат данных из функтора

// The function object multiplies an element by a Factor

template <class Type>

class MultValue

{

private:

Type Factor;   // The value to multiply by

public:

// Constructor initializes the value to multiply by

MultValue(const Type& \_Val) : Factor(\_Val) {

}

// The function call for the element to be multiplied

void operator ( ) (Type& elem) const

{

elem \*= Factor;

}

};

// The function object to determine the average

class Average

{

private:

long num;      // The number of elements

long sum;      // The sum of the elements

public:

// Constructor initializes the value to multiply by

Average() : num(0), sum(0)

{

}

// The function call to process the next elment

void operator ( ) (int elem)

{

num++;      // Increment the element count

sum += elem;   // Add the value to the partial sum

}

// return Average – перегузка оператора приведения типа

operator double()

{

return  static\_cast <double> (sum)

static\_cast <double> (num);

}

};

int main()

{

using namespace std;

vector <int> v1;

vector <int>::iterator Iter1;

// заполнение v1

int i;

for (i = -4; i <= 2; i++)

{

v1.push\_back(i);

}

// вывод v1

cout << "Original vector  v1 = ( ";

for (Iter1 = v1.begin(); Iter1 != v1.end(); Iter1++)

cout << \*Iter1 << " ";

cout << ")." << endl;

// Using for\_each to multiply each element by a Factor

for\_each(v1.begin(), v1.end(), MultValue<int>(-2));

// вывод вектора при помощи ранее использовавшегося итератора после модификации v1 в for\_each

cout << "Multiplying the elements of the vector v1\n "

<< "by the factor -2 gives:\n v1mod1 = ( ";

for (Iter1 = v1.begin(); Iter1 != v1.end(); Iter1++)

cout << \*Iter1 << " ";

cout << ")." << endl;

// умножение вектора на новый коэфициент

// The function object is templatized and so can be

// used again on the elements with a different Factor

for\_each(v1.begin(), v1.end(), MultValue<int>(5));

cout << "Multiplying the elements of the vector v1mod\n "

<< "by the factor 5 gives:\n v1mod2 = ( ";

for (Iter1 = v1.begin(); Iter1 != v1.end(); Iter1++)

cout << \*Iter1 << " ";

cout << ")." << endl;

// The local state of a function object can accumulate

// information about a sequence of actions that the

// return value can make available, here the Average

double avemod2 = for\_each(v1.begin(), v1.end(),

Average());

cout << "The average of the elements of v1 is:\n Average ( v1mod2 ) = "

<< avemod2 << "." << endl;

}

Вывод:

Original vector v1 = ( -4 -3 -2 -1 0 1 2 ).

Multiplying the elements of the vector v1

by the factor -2 gives:

v1mod1 = ( 8 6 4 2 0 -2 -4 ).

Multiplying the elements of the vector v1mod

by the factor 5 gives:

v1mod2 = ( 40 30 20 10 0 -10 -20 ).

The average of the elements of v1 is:

Average ( v1mod2 ) = 10.

Пример 2:

- демонстрация эквивалентности использования статической функции и функтора:

// for\_each example

#include <iostream>     // std::cout

#include <algorithm>    // std::for\_each

#include <vector>       // std::vector

void myfunction(int i)

{  // function:

std::cout << ' ' << i;

}

struct myclass

{           // function object type:

void operator() (int i) { std::cout << ' ' << i; }

} myobject;

int main()

{

std::vector<int> myvector;

myvector.push\_back(10);

myvector.push\_back(20);

myvector.push\_back(30);

std::cout << "myvector contains:";

for\_each(myvector.begin(), myvector.end(), myfunction);

std::cout << '\n';

// or:

std::cout << "myvector contains:";

for\_each(myvector.begin(), myvector.end(), myobject);

std::cout << '\n';

return 0;

}

Пример 3:

The following example uses a [lambda function](http://en.cppreference.com/w/cpp/language/lambda) to increment all of the elements of a vector and then uses an overloaded operator() in a functor to compute their sum:

struct Sum

{

Sum() { sum = 0; }

void operator()(int n) { sum += n; }

int sum;

};

int main()

{

std::vector<int> nums{ 3, 4, 2, 9, 15, 267 };

std::cout << "before: ";

for (auto n : nums)

{

std::cout << n << " ";

}

std::cout << '\n';

std::for\_each(nums.begin(), nums.end(), [](int &n){ n++; });

// Calls Sum::operator() for each number

Sum s = std::for\_each(nums.begin(), nums.end(), Sum());

std::cout << "after:  ";

for (auto n : nums)

{

std::cout << n << " ";

}

std::cout << '\n';

std::cout << "sum: " << s.sum << '\n';

}

### std::reference\_wrapper

<http://www.cplusplus.com/reference/functional/reference_wrapper/>

Class that emulates a reference to an element of type T, but which can be copied (it is both [*copy-constructible*](http://www.cplusplus.com/is_copy_constructible) and[*copy-assignable*](http://www.cplusplus.com/is_copy_assignable)).

### std::(c)ref

<http://www.cplusplus.com/reference/functional/ref/>

Constructs an object of the appropriate [reference\_wrapper](http://www.cplusplus.com/reference_wrapper) type to hold a (const) reference to elem.

### Указатели на функции

#### С++ 98 style:

1) записать тип указателя на функцию, синтаксически воспроизводящий структуру заголовка ф-ии:

а. для статических (свобоных) ф-й:

typedef тип\_возврата (\* имя\_типа\_указателя)(тип\_аргумента1, ..., тип\_аргументаN)

есть ф-ия int foo(double) {…}, тогда у-ль: typedef int (\*pfoo)(double)

б. для ф-й-членов:

тип\_возврата (имя\_класса::\*имя\_типа\_указателя)(тип\_аргумента1, ..., тип\_аргументаN)

есть класс class A{ public: char f(int d) {…} }; , тогда у-ль: typedef char(A::\*ptr\_to\_mem\_func)(int)

2) объявить у-ль заданного типа, инициализировав его конретным у-м на ф-ю:

а. для своб-х ф-й:

имя\_типа\_указателя имя\_указателя = &имя\_функции

pfoo ptrFoo = &foo;

б. для ф-й-членов:

имя\_типа\_указателя имя\_указателя = &имя\_класса::имя\_функции\_члена

ptr\_to\_mem\_func mem\_func\_ptr = &A::f;

**Детали:**

- также указатели на своб-е ф-ии можно связывать со статическими методами классов:

class A

{

public:

static void fn(void);

void fn2(void);

};

typedef void (\*nonmember\_func\_ptr)(void);

typedef void (A::\*member\_func\_ptr)(void);

nonmember\_func\_ptr nfp = &A::fn; // правильно

nonmember\_func\_ptr nfp2 = &A::fn2 // ошибка времени компиляции, A::fn2 – нестатический метод

member\_func\_ptr mfp = &A::fn2; // правильно

3) вызвать функцию по указателю:

а. для свободных ф-й:

int foo(double) {…};

typedef int (\*pfoo)(double);

pfoo ptrFoo = &foo;

int d = ptrFoo(1000);

б. для методов:

class A { public: char f(int d) {…} };

typedef char (A::\* ptr\_to\_mem\_func)(int);

ptr\_to\_mem\_func mem\_func\_ptr = &A::f;

- для вызова метода через указатель на него нужно иметь объект класса, относительно которого будет вызван метод:

объект\_класса.\*имя\_указателя\_на\_метод(аргументы)

указатель\_на\_ объект\_класса->\*имя\_указателя\_на\_метод(аргументы)

A a;

A\* pa;

a.\* mem\_func\_ptr(9000);

pa->\* mem\_func\_ptr(10000);

**Детали:**

- можно обойтись без typedef и сразу связать имя\_указателя с конкретной ф-й:

int (\*pfoo)(double) = &foo;

int d = ptrFoo(1000);

char (A::\* ptr\_to\_mem\_func)(int) = &A::f;

a.\* ptr\_to\_mem\_func (9000);

pa->\* ptr\_to\_mem\_func (10000);

- также для упрощения записи используют макросы:

#define CALL\_MEMBER\_FN(object, ptrToMember) ((object).\*(ptrToMember))

CALL\_MEMBER\_FN(a, ptr\_to\_mem\_func)(5005);

#### C++ 11 style:

##### std::function <functional>

Основные статьи: MSDN function::function <http://msdn.microsoft.com/en-us/library/bb982007.aspx> ,

cppreference std::function <http://en.cppreference.com/w/cpp/utility/functional/function>

- Constructs a wrapper that either is empty or stores a callable object of arbitrary type with a fixed signature

- является функтором – функциональным объектом (объектом, в классе которого перегружен оператор вызова ф-ии)

|  |  |  |
| --- | --- | --- |
| template< class > class function; */\* undefined \*/* |  | (since C++11) |
| template< class R, class... Args > class function<R(Args...)> |  | (since C++11) |
|  |  |  |

Class template std::function is a general-purpose polymorphic function wrapper. Instances of std::function can store, copy, and invoke any [Callable](http://en.cppreference.com/w/cpp/concept/Callable) *target* -- functions, [lambda expressions](http://en.cppreference.com/w/cpp/language/lambda), [bind expressions](http://en.cppreference.com/w/cpp/utility/functional/bind), or other function objects, as well as pointers to member functions and pointers to data members.

The stored callable object is called the *target* of std::function. If a std::function contains no target, it is called *empty*. Invoking the *target* of an *empty* std::function results in [std::bad\_function\_call](http://en.cppreference.com/w/cpp/utility/functional/bad_function_call) exception being thrown.

std::function satisfies the requirements of [CopyConstructible](http://en.cppreference.com/w/cpp/concept/CopyConstructible) and [CopyAssignable](http://en.cppreference.com/w/cpp/concept/CopyAssignable).

**Member types**

|  |  |
| --- | --- |
| **Type** | **Definition** |
| result\_type | R |
| argument\_type | T if sizeof...(Args)==1 and T is the first and only type in Args... |
| first\_argument\_type | T1 if sizeof...(Args)==2 and T1 is the first of the two types in Args... |
| second\_argument\_type | T2 if sizeof...(Args)==2 and T2 is the second of the two types in Args... |

**Member functions**

|  |  |
| --- | --- |
| [(constructor)](http://en.cppreference.com/w/cpp/utility/functional/function/function) | constructs a new std::function instance  (public member function) |
| [(destructor)](http://en.cppreference.com/w/cpp/utility/functional/function/~function) | destroys a std::function instance  (public member function) |
| [**operator=**](http://en.cppreference.com/w/cpp/utility/functional/function/operator%3D) | assigns a new target  (public member function) |
| [**swap**](http://en.cppreference.com/w/cpp/utility/functional/function/swap) | swaps the contents  (public member function) |
| [**assign**](http://en.cppreference.com/w/cpp/utility/functional/function/assign) | assigns a new target  (public member function) |
| [**operator bool**](http://en.cppreference.com/w/cpp/utility/functional/function/operator_bool) | checks if a valid target is contained  (public member function) |
| [**operator()**](http://en.cppreference.com/w/cpp/utility/functional/function/operator()) | invokes the target  (public member function) |
| Target access | |
| [**target\_type**](http://en.cppreference.com/w/cpp/utility/functional/function/target_type) | obtains the typeid of the stored target of a **std::function**  (public member function) |
| [**target**](http://en.cppreference.com/w/cpp/utility/functional/function/target) | obtains a pointer to the stored target of a **std::function**  (public member function) |

**Non-member functions**

|  |  |
| --- | --- |
| [**std::swap**(std::function)](http://en.cppreference.com/w/cpp/utility/functional/function/swap2)  (C++11) | specializes the [std::swap](http://en.cppreference.com/w/cpp/algorithm/swap) algorithm  (function template) |
| [**operator==operator!=**](http://en.cppreference.com/w/cpp/utility/functional/function/operator_cmp) | compares an **std::function** with std::nullptr  (function template) |

**Helper classes**

|  |  |
| --- | --- |
| [**std::uses\_allocator**<std::function>](http://en.cppreference.com/w/cpp/utility/functional/function/uses_allocator)  (C++11) | specializes the [std::uses\_allocator](http://en.cppreference.com/w/cpp/memory/uses_allocator) type trait  (class template specialization) |

Примеры:

Объявление с явным указанием типа:

int func(double r) {…};

std::function<int(double)> pfunc = &func;

pfunc(9.81);

class A {

public:

int func(double) {}

};

- тип указателя задаётся явно:

std::function<int (A&, double)> call\_func1 = &A::func;

std::function<int (A\*, double)> call\_func\_on\_ptr1 = &A::func;

call\_func1(a, 3.1415);

call\_func\_on\_ptr1(pa, 3.1415);

call\_func\_on\_ptr1(&a, 3.1415);

Объявление с неявным заданием типа:

- тип указателя выводится автоматически (3 способа):

а. для статических ф-й:

auto call\_foo = &foo;

decltype(&foo) call\_foo2 = &foo;

std::function<decltype(foo)> call\_foo3 = &foo;

call\_foo(5.0);

call\_foo2(5.0);

call\_foo3(5.0);

б. для методов:

A a;

auto call\_f = std::mem\_fn(&A::f);

call\_f(a, 5.0); // call\_f(&a, 5.0); // call\_f(pa, 5.0);

Пример (MSDN):

typedef std::vector< std::function<int (int)> > vf\_t;

vf\_t v;

v.push\_back(square);

v.push\_back(std::negate<int>());

v.push\_back(multiply\_by(3));

for (vf\_t::const\_iterator i = v.begin(); i != v.end(); ++i)

{

std::cout << (\*i)(10) << std::endl;

}

std::function<int (int)> f = v[0];

std::function<int (int)> g;

##### std::mem\_fn <functional>

Основные статьи: MSDN mem\_fn Function <http://msdn.microsoft.com/en-us/library/bb982851.aspx> ,

cppreference std::mem\_fn <http://en.cppreference.com/w/cpp/utility/functional/mem_fn>

- Generates a simple call wrapper

- возвращает функтор

[C++](http://en.cppreference.com/w/cpp) [Utilities library](http://en.cppreference.com/w/cpp/utility) [Function objects](http://en.cppreference.com/w/cpp/utility/functional)

|  |  |  |
| --- | --- | --- |
| Defined in header [<functional>](http://en.cppreference.com/w/cpp/header/functional) |  |  |
| template< class R, class T > */\*unspecified\*/* mem\_fn(R T::\* pm); | (1) | (since C++11) |
| template< class R, class T, class... Args >  */\*unspecified\*/* mem\_fn(R (T::\* pm)(Args...)); template< class R, class T, class... Args > */\*unspecified\*/* mem\_fn(R (T::\* pm)(Args...) const); template< class R, class T, class... Args >  */\*unspecified\*/* mem\_fn(R (T::\* pm)(Args...) volatile); template< class R, class T, class... Args >  */\*unspecified\*/* mem\_fn(R (T::\* pm)(Args...) const volatile); template< class R, class T, class... Args >  */\*unspecified\*/* mem\_fn(R (T::\* pm)(Args...) &); template< class R, class T, class... Args >  */\*unspecified\*/* mem\_fn(R (T::\* pm)(Args...) const &); template< class R, class T, class... Args >  */\*unspecified\*/* mem\_fn(R (T::\* pm)(Args...) volatile &); template< class R, class T, class... Args >  */\*unspecified\*/* mem\_fn(R (T::\* pm)(Args...) const volatile &); template< class R, class T, class... Args >  */\*unspecified\*/* mem\_fn(R (T::\* pm)(Args...) &&); template< class R, class T, class... Args >  */\*unspecified\*/* mem\_fn(R (T::\* pm)(Args...) const &&); template< class R, class T, class... Args >  */\*unspecified\*/* mem\_fn(R (T::\* pm)(Args...) volatile &&); template< class R, class T, class... Args >  */\*unspecified\*/* mem\_fn(R (T::\* pm)(Args...) const volatile &&); | (2) | (since C++11)  (until C++14) |
|  |  |  |

Function template std::mem\_fn generates wrapper objects for pointers to members, which can store, copy, and invoke a pointer to member. Both references and pointers (including smart pointers) to an object can be used when invoking astd::mem\_fn.

The overloads (2) were introduced in C++11 but removed in C++14 as [defect #2048](http://cplusplus.github.io/LWG/lwg-defects.html#2048)

**Parameters**

|  |  |  |
| --- | --- | --- |
| **pm** | - | pointer to member that will be wrapped |

**Return value**

std::mem\_fn returns a call wrapper of unspecified type that has the following members:

**Member types**

|  |  |
| --- | --- |
| **type** | **definition** |
| result\_type | the return type of pm if pm is a pointer to member function, not defined for pointer to member object |
| argument\_type | T\*, possibly cv-qualified, if pm is a pointer to member function taking no arguments |
| first\_argument\_type | T\* if pm is a pointer to member function taking one argument |
| second\_argument\_type | T1 if pm is a pointer to member function taking one argument of type T1 |

**Member function**

|  |  |
| --- | --- |
| **operator()** | invokes the target on a specified object, with optional parameters  (public member function) |

Пример 1:

- вызов непосредственно при объявлении

class Funcs

{

public:

void square(double x)

{

std::cout << x << "^2 == " << x \* x << std::endl;

}

void product(double x, double y)

{

std::cout << x << "\*" << y << " == " << x \* y << std::endl;

}

};

int main()

{

Funcs funcs;

std::mem\_fn(&Funcs::square)(funcs, 3.0);

std::mem\_fn(&Funcs::product)(funcs, 3.0, 2.0);

return (0);

}

Пример 2:

Use mem\_fn to store and execute a member function and a member object:

- первым передаётся объект, следующими аргументами – параметры метода

struct Foo {

void display\_greeting() {

std::cout << "Hello, world.\n";

}

void display\_number(int i) {

std::cout << "number: " << i << '\n';

}

int data = 7;

};

int main() {

Foo f;

auto greet = std::mem\_fn(&Foo::display\_greeting);

greet(f);

auto print\_num = std::mem\_fn(&Foo::display\_number);

print\_num(f, 42);

auto access\_data = std::mem\_fn(&Foo::data);

std::cout << "data: " << access\_data(f) << '\n';

}

Пример 3:

- передача метода в ф-ию без указания объекта вызова

Pass a member function to [std::transform](http://en.cppreference.com/w/cpp/algorithm/transform) to create a sequence of numbers:

#include <iostream>

#include <functional>

#include <iterator>

#include <memory>

#include <string>

#include <vector>

#include <algorithm>

int main()

{

std::vector<std::string> words = { "This", "is", "a", "test" };

std::vector<std::unique\_ptr<std::string>> words2;

words2.emplace\_back(new std::string("another"));

words2.emplace\_back(new std::string("test"));

std::vector<std::size\_t> lengths;

std::transform(words.begin(),

words.end(),

std::back\_inserter(lengths),

std::mem\_fn(&std::string::size)); // uses references to strings

std::transform(words2.begin(),

words2.end(),

std::back\_inserter(lengths),

std::mem\_fn(&std::string::size)); // uses unique\_ptr to strings

std::cout << "The string lengths are ";

for (auto n : lengths) std::cout << n << ' ';

std::cout << '\n';

}

Пример 4:

- C++14 и разрешение перегрузки

Demonstrates the effect of the C++14 changes to the specification of std::mem\_fn

struct X {

int x;

int&       easy()      { return x; }

int&       get()       { return x; }

const int& get() const { return x; }

};

int main(void)

{

auto a = std::mem\_fn(&X::easy); // no problem at all

//  auto b = std::mem\_fn<int&  >(&X::get ); // no longer works in C++14

auto c = std::mem\_fn<int&()>(&X::get); // works with both C++11 and C++14

auto d = [](X& x) {return x.get(); };   // another approach to overload resolution

X x = { 33 };

std::cout << "a() = " << a(x) << '\n';

std::cout << "c() = " << c(x) << '\n';

std::cout << "d() = " << d(x) << '\n';

}

Вывод:

a() = 33

c() = 33

d() = 33

##### std::bind <functional>

Основные статьи: cppreference std::bind <http://ru.cppreference.com/w/cpp/utility/functional/bind> , MSDN bind Function <http://msdn.microsoft.com/en-us/library/bb982702.aspx>

- Binds arguments to a callable object

- возвращает функтор

template <class Fn, class... Args>

/\* unspecified \*/ bind (Fn&& fn, Args&&... args);

template <class Ret, class Fn, class... Args>

/\* unspecified \*/ bind (Fn&& fn, Args&&... args);

|  |  |  |
| --- | --- | --- |
| template< class F, class... Args > */\*unspecified\*/* bind( F&& f, Args&&... args ); | (1) | (since C++11) |
| template< class R, class F, class... Args > */\*unspecified\*/* bind( F&& f, Args&&... args ); | (2) | (since C++11) |
|  |  |  |

The function template bind generates a forwarding call wrapper for f. Calling this wrapper is equivalent to invoking fwith some of its arguments bound to args.

**Parameters**

|  |  |  |
| --- | --- | --- |
| **f** | - | [Callable](http://en.cppreference.com/w/cpp/concept/Callable) object (function object, pointer to function, reference to function, pointer to member function, or pointer to data member) that will be bound to some arguments |
| **args** | - | list of arguments to bind, with the unbound arguments replaced by the placeholders \_1, \_2, \_3... of namespace std::placeholders |

**Return value**

A function object of unspecified type T, for which [std::is\_bind\_expression](http://en.cppreference.com/w/cpp/utility/functional/is_bind_expression)<T>::value == true, and which can be stored in [std::function](http://en.cppreference.com/w/cpp/utility/functional/function). It has the following members:

**std::bind *return type***

**Member objects**

The return type of std::bind holds a member object of type [std::decay](http://en.cppreference.com/w/cpp/types/decay)<F>::type constructed from [std::forward](http://en.cppreference.com/w/cpp/utility/forward)<F>(f), and one object per each of args..., of type [std::decay](http://en.cppreference.com/w/cpp/types/decay)<Arg\_i>::type, similarly constructed from [std::forward](http://en.cppreference.com/w/cpp/utility/forward)<Arg\_i>(arg\_i).

**Constructors**

The return type of std::bind is [CopyConstructible](http://en.cppreference.com/w/cpp/concept/CopyConstructible) if all of its member objects (specified above) are CopyConstructible, and is [MoveConstructible](http://en.cppreference.com/w/cpp/concept/MoveConstructible) otherwise. The type defines the following members:

**Member type result\_type**

1) If F is a pointer to function or a pointer to member function, result\_type is the return type of F. If F is a class type with nested typedef result\_type, then result\_type is F::result\_type. Otherwise no result\_type is defined.

2) result\_type is exactly R.

**Member function operator()**

Given an object g obtained from an earlier call to bind, when it is invoked in a function call expression g(u1, u2, ... uM), an invocation of the stored object of type [std::decay](http://en.cppreference.com/w/cpp/types/decay)<F>::type takes place, with arguments defined as follows:

* If the argument is of type std::reference\_wrapper<T> (for example, [std::ref](http://en.cppreference.com/w/cpp/utility/functional/ref) or [std::cref](http://en.cppreference.com/w/cpp/utility/functional/ref) was used in the initial call to bind), then the reference T& stored in the bound argument is passed to the invocable object.
* If [std::is\_bind\_expression](http://en.cppreference.com/w/cpp/utility/functional/is_bind_expression)<T>::value == true (i.e. another bind subexpression was used as an argument in the initial call to bind), then that bind subexpression is invoked immediately and its result is passed to the invocable object. If the bind subexpression has any placeholder arguments, they are picked fromu1, u2, ....
* If [std::is\_placeholder](http://en.cppreference.com/w/cpp/utility/functional/is_placeholder)<T>::value != 0 (i.e., std::placeholders::\_1, \_2, \_3, ... was used as the argument to the initial call to bind), then the argument indicated by the placeholder (u1 for \_1, u2 for \_2, etc) is passed to the invocable object as [std::forward](http://en.cppreference.com/w/cpp/utility/forward)<Uj>(uj).
* Otherwise, the stored argument is passed to the invocable object as-is.

If some of the arguments that are supplied in the call to g() are not matched by any placeholders stored in g, the unused arguments are evaluated and discarded.

**Exceptions**

Only throws if construction of [std::decay](http://en.cppreference.com/w/cpp/types/decay)<F>::type from [std::forward](http://en.cppreference.com/w/cpp/utility/forward)<F>(f) throws, or any of the constructors for [std::decay](http://en.cppreference.com/w/cpp/types/decay)<Arg\_i>::type from the corresponding [std::forward](http://en.cppreference.com/w/cpp/utility/forward)<Arg\_i>(arg\_i) throws where Arg\_i is the ith type and arg\_i is the ith argument in Args... args.

**Notes**

As described in [Callable](http://en.cppreference.com/w/cpp/concept/Callable), when invoking a pointer to non-static member function or pointer to non-static data member, the first argument has to be a reference or pointer (including, possibly, smart pointer such as[std::shared\_ptr](http://en.cppreference.com/w/cpp/memory/shared_ptr) and [std::unique\_ptr](http://en.cppreference.com/w/cpp/memory/unique_ptr)) to an object whose member will be accessed.

The arguments to bind are copied or moved, and are never passed by reference unless wrapped in [std::ref](http://en.cppreference.com/w/cpp/utility/functional/ref) or[std::cref](http://en.cppreference.com/w/cpp/utility/functional/ref).

Duplicate placeholders in the same bind expression (multiple \_1's for example) are allowed, but the results are only well defined if the corresponding argument (u1) is an lvalue or non-movable rvalue.

- создаёт и возвращает объект-функцию ОФ (т.е. может быть сохранён в std::function), связанный с fn: функцией, методом или полем; при этом сущность, переданная в fn может быть связана с произвольным числом параметров args:

###### std::bind и функции

- на базе ф-ии fn можно создать объект-ф-ию (ОФ), связанный с конкретными значениями, которые будут всегда неявно передаваться в соответ-ие аргументы во время вызова ОФ, где соотв-ие задаётся как простым порядком указания связываемых значений, так и специальными местозаполнителями (std::placeholder::\_1, …, std::placeholder::\_N – где N любое натуральное число);

-сами связываемые значения будут копироваться либо перемещаться в связанную ф-ию, и для того чтобы они передавались по ссылке или константной ссылке нужно использовать оболчку std::ref или std::cref

double fn\_divide (double x, double y) {return x/y;}

struct TwoInts {

double a,b;

double multiply() {return a\*b;}

};

using namespace std::placeholders; // adds visibility of \_1, \_2, \_3,...

// binding functions:

auto fn\_five = std::bind (fn\_divide,10,2); // returns 10/2

std::cout << fn\_five() << '\n'; // 5

auto fn\_five = std::bind (fn\_divide,\_1,\_1); // returns 10/10

std::cout << fn\_five(10) << '\n'; // 1

auto fn\_half = std::bind (fn\_divide,\_1,2); // returns x/2

std::cout << fn\_half(10) << '\n'; // 5

auto fn\_invert = std::bind (fn\_divide,\_2,\_1); // returns y/x

std::cout << fn\_invert(10,2) << '\n'; // 0.2 - здесь за счёт

место заполнителей

поменяли явный порядок передачи аргументов

auto fn\_rounding = std::bind<int> (fn\_divide,\_1,\_2); // returns int(x/y)

std::cout << fn\_rounding(10,3) << '\n'; // 3 - здесь передали тип возвращаемого значения – int, поэтому мы получили выражение int(10/3) = 3

double d = 10.0;

auto fn\_five = std::bind (fn\_divide,std::ref(d),2); // returns 10/2

std::cout << fn\_five() << '\n'; // 5 – передали d в ф-ию по ссылке, т.о. его можно там изменить

Ещё примеры:

void f(int n1, int n2, int n3, const int& n4, int n5)

{

std::cout << n1 << ' ' << n2 << ' ' << n3 << ' ' << n4 << ' ' << n5 << '\n';

}

int g(int n1)

{

return n1;

}

// demonstrates argument reordering and pass-by-reference

int n = 7;

// (\_1 and \_2 are from std::placeholders, and represent future

// arguments that will be passed to f1)

auto f1 = std::bind(f, \_2, \_1, 42, std::cref(n), n);

n = 10;

f1(1, 2, 1001); // первый параметр, передаваемый в f1 будет передан во второй параметр ф-ии f,

// второй – в первый, 1001 – не используется и будет отброшен;

Вывод: 2 1 42 10 7

// здесь вложенные bind выражения разделяют местозаполнители

auto f2 = std::bind(f, \_3, std::bind(g, \_3), \_3, 4, 5);

f2(10, 11, 12); // в данном случае используется только 3-й фактический параметр (т.к. это

// задано местозаполнителем \_3), т.е. 12, который будет подставлен вместо первого параметра ф-ии f, 10 и 11 будут отброшены; вторым параметром для f будет рез-т выражения std::bind(g, \_3), т.е. рез-т передачи в ф-ю g параметра, связанного с местозаполнителем \_3, т.е. g(12); в третий параметр f будет передано опять значение 12; в 4-й и 5-й параметры, будут переданы 4 и 5, соотв-но

Вывод: 12 12 12 4 5

// common use case: binding a RNG with a distribution

std::default\_random\_engine e; // создаём генератор случайных чисел

std::uniform\_int\_distribution<> d(0, 10); // создаём ф-ию, задающую распределение с.ч., и диапазон значений, в котором они будут генерироваться (область определения для ф-ии распределения), в данном случае это отрезок [0, 10], т.е. от 0 до 10 включительно

std::function<int()> rnd = std::bind(d, e); // создаём ОФ на базе распределения с.ч. d, в оператор вызова ф-ии operator(), которого передаём генератор с.ч. e

// генерируем распределение 10 раз

for (int n = 0; n<10; ++n)

std::cout << rnd() << ' ';

std::cout << '\n';

Вывод: 1 5 0 2 0 8 2 2 10 8

###### std::bind и методы

- дополнительно к тому что относится к ф-м, результирующий ОФ можно, связать с конкретным экземпляром класса, на котором требуется сделать вызов связанного метода

TwoInts ten\_two {10,2};

auto bound\_member\_fn = std::bind (&TwoInts::multiply,\_1); // returns x.multiply()

std::cout << bound\_member\_fn(ten\_two) << '\n'; // 20

т.е. в данном случае параметризуется объект, на котором будет вызываться метод; \_1 означает, что объект (точнее его копия, т.к. не использовались std::ref и std::cref), переданный в первом параметре bound\_member\_fn будет использован для вызова метода TwoInts::multiply

class A

{

public:

int foo(double d1, int n1, int n2)

{

std::cout << d1 << "  " << n1 << "  " << n2 << "\n\n";

return 0;

}

float data = 55555;

};

A a;

auto custom\_foo = std::bind(&A::foo, &a, 1000, \_1, \_2);

custom\_foo(4000, 5000);

Вывод: 1000, 4000, 5000

###### std::bind и поля

- в параметре args можно передать объект, на котором будет получен доступ к полю, переданному в параметре fn

auto bound\_member\_data = std::bind (&TwoInts::a,ten\_two); // returns ten\_two.a

std::cout << bound\_member\_data() << '\n'; // 10

A a;

auto get\_field\_on\_custom\_obj = std::bind(&A::data, \_1);

cout << get\_field\_on\_custom\_obj(a) << "\n\n"; // 55555

###### std::bind и умные указатели

- чтобы передать указатель на объект, на котором будет вызываться член класса, можно использовать умные указатели std::shared\_ptr и std::unique\_ptr

cout << get\_field\_on\_custom\_obj(std::make\_shared<A>(a)) << "\n\n"; // 55555

cout << get\_field\_on\_custom\_obj(std::unique\_ptr<A>(new A(a))) << "\n\n"; // 55555

cout << get\_field\_on\_custom\_obj(std::make\_unique<A>(a)) << "\n\n"; // 55555

###### std::bind и std::for\_each

Пример 1:

void square(double x)

{

std::cout << x << "^2 == " << x \* x << std::endl;

}

void product(double x, double y)

{

std::cout << x << "\*" << y << " == " << x \* y << std::endl;

}

double arg[] = { 1, 2, 3 };

// Для каждого элемента массива arg вызываются ф-ии square и product.

std::for\_each(&arg[0], arg + 3, square);

std::cout << std::endl;

std::for\_each(&arg[0], arg + 3, std::bind(product, \_1, 2));

std::cout << std::endl;

std::for\_each(&arg[0], arg + 3, std::bind(square, \_1));

Пример 2:

vector<int> v = { 1, 2, 3, 4, 5 };

A a;

auto custom\_foo = bind(&A::foo, a, 3.1415, 55555, \_1);

for\_each(v.begin(), v.end(), custom\_foo);

или

for\_each(v.begin(), v.end(), bind(&A::foo, a, 3.1415, 55555, \_1));

Вывод:

3.1415 55555 1

3.1415 55555 2

3.1415 55555 3

3.1415 55555 4

3.1415 55555 5

Пример 3:

class A

{

public:

A(double n) : data(n)

{

}

int foo(double d1, int n1, int n2)

{

std::cout << d1 << "  " << n1 << "  " << n2 << "  " << data << "\n\n";

return 0;

}

int method1()

{

std::cout << data << "\n\n";

return 0;

}

double data = 55555;

};

void main()

{

vector<A> v = { 1, 2, 3, 4, 5 };

for\_each(v.begin(), v.end(), std::bind(&A::foo, \_1, 3.1415, 55555, 77777)); // из вектора извлекаются объекты класса A, подставляются вместо \_1 и на каждом из них вызывается метод foo, остальные аргументы жёстко заданы

auto custom\_foo\_method = bind(&A::foo, 3.1415, 55555, 77777);

for\_each(v.begin(), v.end(), std::mem\_fn(&A::method1));

}

- the syntax would be the same if a\_vect was a std::vector<A\*>

- these 2 statements transform a function that is called on an object to a function that takes the object it is supposed to be called on as a parameter.

Вывод:

3.1415 55555 77777 1

3.1415 55555 77777 2

3.1415 55555 77777 3

3.1415 55555 77777 4

3.1415 55555 77777 5

1

2

3

4

5

### Функциональные объекты - функторы

<http://msdn.microsoft.com/en-us/library/aa985932.aspx> Function Objects

- ФО – всякий класс, перегружающий оператор вызова ф-ии

- используются в качестве предикатов к алгоритмам STL и как компараторы контейнеров STL, хранящих отсортированные последовательности значений

- позволяют хранить внутренне состояние (allow to maintain state)

A *function object*, or *functor*, is any type that implements operator(). This operator is referred to as the *call operator* or sometimes the *application operator*. The Standard Template Library uses function objects primarily as sorting criteria for containers and in algorithms.

Function objects provide two main advantages over a straight function call. The first is that a function object can contain state. The second is that a function object is a type and therefore can be used as a template parameter.

To create a function object, create a type and implement operator(), such as:

class Functor

{

public:

int operator()(int a, int b)

{

return a < b;

}

};

int main()

{

Functor f;

int a = 5;

int b = 7;

int ans = f(a, b);

}

#### Function Objects and Containers

The Standard Template Library contains several function objects in the [<functional>](http://msdn.microsoft.com/en-us/library/169de7b0.aspx) header file. One use of these function objects is as a sorting criterion for containers. For example, the **set** container is declared as follows:

template <

class Key,

class Traits = less<Key>,

class Allocator = allocator<Key> >

class set

The second template argument is the function object **less**. This function object returns **true** if the first parameter passed to it is less than the second parameter passed. Since some containers sort their elements, the container needs a way of comparing two elements, and this is accomplished using the function object. You can define your own sorting criteria for containers by creating a function object and specifying it in the template list for the container.

#### Function Objects and Algorithms

Another use of functional objects is in algorithms. For example, the **remove\_if** algorithm is declared as follows:

template<class ForwardIterator, class Predicate>

ForwardIterator remove\_if(

ForwardIterator \_First,

ForwardIterator \_Last,

Predicate \_Pred

);

The last argument to **remove\_if** is a function object that returns a boolean value (a *predicate*). If the result of the function object is **true**, then the element is removed from the container being accessed by the iterators *\_First* and *\_Last*. You can use any of the function objects declared in the [<functional>](http://msdn.microsoft.com/en-us/library/169de7b0.aspx) header for the argument *\_Pred* or you can create your own.

### Лямбда выражения (лямбда функции, анонимные функции, замыкания)

<http://msdn.microsoft.com/en-us/library/dd293608.aspx> Lambda Expressions in C++

<http://msdn.microsoft.com/en-us/library/dd293603.aspx> Lambda Expression Syntax

<http://www.dreamincode.net/forums/topic/264061-c11-fun-with-functions/>

<http://en.cppreference.com/w/cpp/language/lambda>

- функции, не имеющие имени

- замыкание – безымянный функтор, способный захватывать переменные в контексте

- сочетают в себе преимущества указателей на ф-ии и функторов - краткость синтаксиса первых и возможность хранить внутреннее состояние вторых

- недостатки указателей на ф-ии и функторов: первые не позволяют хранить внутреннее состояние, вторые требуют объявления класса

- using lambdas, you can write code that's less cumbersome and less prone to errors than the code for an equivalent function object

- создаётся безымянный ФО (как и для std::function/bind/mem\_fn и др.)

- the lambda expression constructs an unnamed rvalue temporary object of unique unnamed non-union non-aggregate type, known as *closure type*, which is declared (for the purposes of [ADL](http://en.cppreference.com/w/cpp/language/adl) – argument dependent lookup) in the smallest block scope, class scope, or namespace scope that contains the lambda expression

[**Лямбда-выражение**](https://ru.wikipedia.org/wiki/%D0%9B%D1%8F%D0%BC%D0%B1%D0%B4%D0%B0-%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F) (в программировании) — это специальный синтаксис для объявления [анонимных функторов](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%82%D0%BE%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) по месту их использования. Используя лямбда-выражения, можно объявлять функции в любом месте кода. Обычно лямбда-выражение допускает [замыкание](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BC%D1%8B%D0%BA%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) на [лексический контекст](https://ru.wikipedia.org/w/index.php?title=%D0%9B%D0%B5%D0%BA%D1%81%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%BA%D1%81%D1%82&action=edit&redlink=1), в котором это выражение использовано.

Лямбда-выражения поддерживаются во многих языках программирования ([C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [Common Lisp](https://ru.wikipedia.org/wiki/Common_Lisp), [Python](https://ru.wikipedia.org/wiki/Python), [PHP](https://ru.wikipedia.org/wiki/PHP), [C#](https://ru.wikipedia.org/wiki/C_Sharp), [F#](https://ru.wikipedia.org/wiki/F_Sharp), [Visual Basic .NET](https://ru.wikipedia.org/wiki/Visual_Basic_.NET), [C++](https://ru.wikipedia.org/wiki/C%2B%2B), [Scala, Java](https://ru.wikipedia.org/wiki/Java) и других).

[**Замыкание**](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BC%D1%8B%D0%BA%D0%B0%D0%BD%D0%B8%D0%B5_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *closure*) в программировании — [функция](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющиеся её параметрами. Говоря другим языком, замыкание — функция, которая ссылается на [свободные переменные](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F) в своём [контексте](https://ru.wikipedia.org/w/index.php?title=%D0%9B%D0%B5%D0%BA%D1%81%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%BA%D1%81%D1%82&action=edit&redlink=1).

Замыкание, так же как и [экземпляр объекта](https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D0%B7%D0%B5%D0%BC%D0%BF%D0%BB%D1%8F%D1%80_%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%B0), есть способ представления функциональности и данных, связанных и упакованных вместе.

Замыкание — это особый вид функции. Она определена в теле другой функции и создаётся каждый раз во время её выполнения. Синтакстически это выглядит как функция, находящаяся целиком в теле другой функции. При этом вложенная внутренняя функция содержит ссылки на локальные переменные внешней функции. Каждый раз при выполнении внешней функции происходит создание нового экземпляра внутренней функции, с новыми ссылками на переменные внешней функции.

В случае замыкания ссылки на переменные внешней функции действительны внутри вложенной функции до тех пор, пока работает вложенная функция, даже если внешняя функция закончила работу, и переменные вышли из области видимости.[[1]](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BC%D1%8B%D0%BA%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)#cite_note-1)

Замыкание связывает код функции с её лексическим окружением (местом, в котором она определена в коде). Лексические переменные замыкания отличаются от глобальных переменных тем, что они не занимают глобальное пространство имён. От переменных в объектах они отличаются тем, что привязаны к функциям, а не объектам.

[**Свободная переменная**](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F) — [переменная](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), которая:

* встречается в теле [функции](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) или [предложения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_(%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B0)), но которая не является параметром этой функции,
* и/или определена в месте, находящемся где-то за пределами функции.

Другими словами, если есть переменная, объявленная где-то в программе и есть функция, которая имеет доступ к этой переменной, то такая переменная будет называться свободной. Иногда её называют глобальной переменной.

#### Синтаксис:

 ISO C++11:

*lambda-introducer lambda-declarator*opt *compound-statement*

*lambda-introducer:*   
        **[** *lambda-capture*opt **]**  
*lambda-capture:*  
        *capture-default*  
        *capture-list*  
        *capture-default* **,** *capture-list*  
*capture-default:*  
        **&**  
        **=**  
*capture-list:*  
        *capture ...*opt  
        *capture-list* **,** *capture ...*opt  
*capture:*  
        *identifier*  
        **&** *identifier*  
        **this**  
*lambda-declarator:*  
        **(** *parameter-declaration-clause* **)** **mutable**opt  
                *exception-specification*opt *attribute-specifier-seq*opt *trailing-return-type*opt

или в виде графа:

*lambda-introducer:* 

**[** *lambda-capture*opt **]**

*lambda-capture:*

**&** *identifier*

*identifier*

**=**

**&**

*capture-default*

**this**

*capture ...*opt

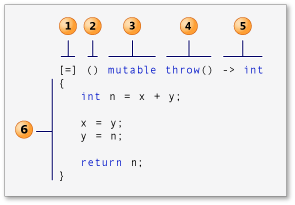
*capture-list* **,** *capture ...*opt

*capture-list*

*capture-default* **,** *capture-list*

*lambda-declarator*opt *:*

**(** *parameter-declaration-clause* **)** **mutable**opt   *exception-specification*opt *attribute-specifier-seq*opt *trailing-return-type*opt



1. *lambda-introducer* (Also known as the *capture clause*)
2. *lambda declarator* (Also known as the *parameter list*)
3. *mutable* (Also known as the *mutable specification*)
4. *exception-specification* (Also known as the *exception specification*)
5. *trailing-return-type* (Also known as the *return type*)
6. *compound-statement* (Also known as the *lambda body*)

Специфика VS и C++11:

Не поддерживаются:

* Like all other classes, lambdas don't get automatically generated move constructors and move assignment operators. For more information about support for rvalue reference behaviors, see the "Rvalue References" section in [Support For C++11 Features (Modern C++)](http://msdn.microsoft.com/en-us/library/hh567368.aspx).
* The optional *attribute-specifier-seq* is not supported in this version.

Возможности сверх спецификации стандарта:

* Stateless lambdas, which are omni-convertible to function pointers that use arbitrary calling conventions (лямбды, в которых нет переменных, хранящих состояние ф-ии)
* Automatically deduced return types for lambda bodies that are more complicated than { return expression; }, as long as all return statements have the same type. (This functionality is part of the proposed C++14 Standard.) Типы возврата выводятся автоматически для одного пути возврата, для множественных путей возрата - только когда все возвращаемые выражения имеют один результирующий тип.

#### Capture clause (lambda-introducer):

Capture clause служит для захвата свободных переменных (переменных, находящихся в контексте объявления ЛФ). Захват может происходить по ссылке и по значению, можно указать захватывать как все внешние переменные, которые будут использованы в теле ЛФ; так и конкретные, указав их идентификаторы.

[]: no variables captured. Attempting to use any external variables in the lambda is an error.

[x, &y]: x is captured by value, y is captured by reference

[&]: any external variable is implicitly captured by reference if used

[=]: any external variable is implicitly captured by value if used

[&, x]: x is explicitly captured by value. Other variables will be captured by reference

[=, &z]: z is explicitly captured by reference. Other variables will be captured by value

- default capture mode позволяет захватывать не указанные явно переменные по ссылке или по значению, для чего в качестве первого символа capture clause используются & или =

- при использовании default capture mode захватываются только те переменные, которые используются в теле ЛФ

- любой идентификатор может быть указан в capture clause один раз

- лямбда, написанная в теле метода, может захватить this и т.о. получить доступ к методам и данным класса: see "Example: Using a Lambda Expression in a Method" in [Examples of Lambda Expressions](http://msdn.microsoft.com/en-us/library/dd293599.aspx) :

class Scale

 {

 public:

     // The constructor.

     explicit Scale(int scale) : \_scale(scale) {}

     // Prints the product of each element in a vector object

     // and the scale value to the console.

     void ApplyScale(const vector<int>& v) const

     {

         for\_each(v.begin(), v.end(), [=](int n) { cout << n \* \_scale << endl; });

или

         //for\_each(v.begin(), v.end(), [this](int n) { cout << n \* \_scale << endl; });

     }

 private:

     int \_scale;

 };

- захватывать можно и пакеты аргументов вариативных шаблонов, a *capture* followed by an ellipsis is a pack expansion:

template<class... Args>

void f(Args... args)

{

auto x = [args...] { return g(args...); };

x();

}

- this и default capture mode не могут использоваться вместе:

[=, this]{}; // ERROR: this when = is the default

#### Parameter list (lambda-declarator):

- является опциональным

- в качестве аргумента может быть передана другая лямбда-функция: see "Higher-Order Lambda Expressions" in the topic [Examples of Lambda Expressions](http://msdn.microsoft.com/en-us/library/dd293599.aspx)

- поскольку список параметров является опциональным, то можно опустить пустые скобки, если не используются:

- *exception-specification*

- *trailing-return-type*

- mutable

#### Mutable specification:

- позволяет модифицировать захваченные по значению внешние переменные

- оператор вызова ф-ии для неявно создаваемого класса ЛФ определяется как const, т.о. свободные переменные, захватываемые по значению, не могут быть изменены в теле ЛФ

- для возможности модифицировать внешние переменные, захватываемые по значению, внутри ЛФ её нужно объявлять с ключевым словом mutable

-  It does not produce mutable data members

vector<int> v1{ 1, 2, 3 };

vector<int> v2{ 4, 5, 6 };

int i = 0;

for\_each(v2.begin(), v2.end(), [&v1, i](int e)mutable{ v1[i] = e; ++i; });

for (auto e : v1)

cout << e << endl;

cout << endl;

#### Exception specification

- можно использовать спецификацию пустого исключения throw(), чтобы показать, что лямбда выражение (ЛВ) не бросает никаких исключений

- также как и для обычных ф-й компилятор сгенерирует предупреждение [C4297](http://msdn.microsoft.com/en-us/library/84e2zwhh.aspx) , если ЛВ объявлено со спецификацией исключения throw(), а тело лямбды бросает исключение:

// compile with: /W4 /EHsc

int main() // C4297 expected

{

[]() throw() { throw 5; }();

}

#### Return type

- for omitted trailing-return-type: the return type of the closure's operator() is determined according to the following rules:

(until C++14)

* if the *body* consists of nothing but a single return statement with an expression, the return type is the type of the returned expression (after [rvalue-to-lvalue, array-to-pointer, or function-to-pointer implicit conversion](http://en.cppreference.com/w/cpp/language/implicit_cast#Lvalue_transformations))
* otherwise, the return type is void

The return type is [deduced](http://en.cppreference.com/w/cpp/language/template_argument_deduction) from [return](http://en.cppreference.com/w/cpp/language/return) statements as if for a function whose [return type is declared auto](http://en.cppreference.com/w/cpp/language/function#Return_type_deduction) (since C++14)

- тип возврата выводится автоматически, если:

- не указан ведущий тип возврата (trailing return type)

- if the lambda body contains just one return statement or the expression does not return a value

- для множественных путей возрата - только когда все возвращаемые выражения имеют один

результирующий тип

- если компилятор не может автоматически вывести тип возврата, то он определяется как void

auto x2 = []{ return{ 1, 2 }; }; // ERROR: return type is void, deducing

// return type from braced-init-list is not valid

- a lambda expression can produce another lambda expression as its return value. For more information, see "Higher-Order Lambda Expressions" in [Examples of Lambda Expressions](http://msdn.microsoft.com/en-us/library/dd293599.aspx).

#### Lambda body (compound-statement)

-  the body of a lambda expression can access:

- class data members, when declared inside a class and this is captured

- any variable that has static storage duration (no capture needed), for example, global variables:

заполнение вектора значениями, которые возвращает ЛФ

// A local static variable.

    static int nextValue = 1;

    // The lambda expression that appears in the following call to

    // the generate function modifies and uses the local static

    // variable nextValue.

    generate(v.begin(), v.end(), [] { return nextValue++; });

#### Рекурсия

std::function<int(int)> fact = [&fact] (int n)

        {

                return (n==0) ? 1 : n \* fact(n-1);

        };

- в этом случае для объявления fact вместо auto используется std::function

#### Higher-order lambda functions

- A higher-order lambda function is a lambda expression that takes another lambda expression as its argument or returns a lambda expression

- You can use the [function](http://msdn.microsoft.com/en-us/library/bb982519.aspx) class to enable a C++ lambda expression to behave like a higher-order function

void main()

{

    // The following code declares a lambda expression that returns

    // another lambda expression that adds two numbers.

    // The returned lambda expression captures parameter x by value.

    auto addtwointegers = [](int x) -> function<int(int)> {

        return [=](int y) { return x + y; };

    };

    // The following code declares a lambda expression that takes another

    // lambda expression as its argument.

    // The lambda expression applies the argument z to the function f

    // and multiplies by 2.

    auto higherorder = [](const function<int(int)>& f, int z) {

        return f(z) \* 2;

    };

    // Call the lambda expression that is bound to higherorder.

    auto answer = higherorder(addtwointegers(7), 8);

    // Print the result, which is (7+8)\*2.

    cout << answer << endl;

}

#### Синтаксис объявления и вызова лямбда-функций

//1 – сохранение ЛФ в переменной

auto double\_n = [](double x) {return x \* 2;};

auto addtwointegers = [](int x) -> function<int(int)> {

        return [=](int y) { return x + y; };

    };

//2 – объявление с одновременным вызовом

int m = 0;

int n = 0;

[&, n] (int a) mutable { m = ++n + a; }(4);

//3 – объявление ЛФ при передаче её в качестве функционального объекта в другую ф-ю

std::sort(v.begin(), v.end(), [](int a, int b){return a < b;});

//4 – сохранение ЛФ в std::function

std::function<bool(int a, int b)> function\_less\_to\_larger = [](int a, int b){return a < b;};

### Применение функторов, указателей на функции, лямбда функций, std::funсtion, std::bind и std::mem\_fn в алгоритмах STL

bool sort\_less\_to\_larger\_predicate(int a, int b)

{

return a < b;

}

void outp\_int(int val)

{

cout << val << " ";

}

class sort\_larger\_to\_less\_predicate\_functor

{

public:

bool operator()(int a, int b)

{

return a > b;

}

};

std::vector<int> v{ -5, -100, 3, 1, 0, 2, 38, -2, 4 };

    /\* // Для VS 2012

    int vals[] = { -5, -100, 3, 1, 0, 2, 38, -2, 4 };

    std::vector<int> v(vals, vals+sizeof(vals)/sizeof(vals[0]));

    \*/

    auto sort\_vector = [&](const std::function<bool(int, int)>& f)

    {

        std::sort(v.begin(), v.end(), f);

    };

    auto outp\_vector = [&]()

    {

        std::for\_each(v.begin(), v.end(), outp\_int);

        cout << "\n\n";

    };

    // ф-ия

    sort\_vector(sort\_less\_to\_larger\_predicate);

    outp\_vector();

    // указатель на ф-ию

    typedef bool (\*Sort\_less\_to\_larger\_predicate\_function\_pointer)(int, int);

Sort\_less\_to\_larger\_predicate\_function\_pointer sort\_less\_to\_larger\_predicate\_function\_pointer = sort\_less\_to\_larger\_predicate;

    sort\_vector(sort\_less\_to\_larger\_predicate\_function\_pointer);

    outp\_vector();

    // ф-й объект

    sort\_vector(sort\_larger\_to\_less\_predicate\_functor());

    outp\_vector();

    // лямбда ф-ия

    sort\_vector([](int a, int b){return a < b;});

    outp\_vector();

    // std::function

    std::function<bool(int a, int b)> larger\_to\_less\_stdfunction = [](int a, int b){return a > b;};

    sort\_vector(larger\_to\_less\_stdfunction);

    outp\_vector();

    // std::bind

    auto sort\_less\_to\_larger\_predicate\_bind = std::bind(sort\_less\_to\_larger\_predicate, \_1, \_2);

    sort\_vector(sort\_less\_to\_larger\_predicate\_bind);

    outp\_vector();

    // std::mem\_fn

auto sort\_larger\_to\_less\_memfn = std::mem\_fn(&sort\_larger\_to\_less\_predicate\_functor::operator());

    sort\_vector(sort\_larger\_to\_less\_memfn);

## Шаблоны

- If we want to define a different implementation for a template when a specific type is passed as template

parameter, we can declare a specialization of that template.

- syntax:

template<> class identifier <char> { ... };

## STL

Стандартная Библиотека Шаблонов предоставляет набор хорошо сконструированных и согласованно работающих вместе обобщённых компонентов C++. **Особая забота была проявлена для обеспечения того, чтобы все шаблонные алгоритмы работали не только со структурами данных в библиотеке, но также и с встроенными структурами данных C++. Например, все алгоритмы работают с обычными указателями.** **Ортогональный проект библиотеки позволяет программистам использовать библиотечные структуры данных со своими собственными алгоритмами, а библиотечные алгоритмы - со своими собственными структурами данных.** Хорошо определённые семантические требования и требования сложности гарантируют, что компонент пользователя будет работать с библиотекой и что он будет работать эффективно. Эта гибкость обеспечивает широкую применимость библиотеки.

    Другое важное соображение - эффективность. C++ успешен, потому что он объединяет выразительную мощность с эффективностью. Много усилий было потрачено, чтобы проверить, что каждый шаблонный компонент в библиотеке имеет обобщённую реализацию, которая имеет эффективность выполнения с разницей в пределах нескольких процентов от эффективности соответствующей программы ручной кодировки.

    Третьим соображением в проекте была разработка библиотечной структуры, которая, будучи естественной и лёгкой для понимания, основана на прочной теоретической основе.

- основная концепция STL – разделение данных и операций над ними; данные хранятся в контейнерах, а оперции представлены обобщёнными алгоритмами; итераторы склеивают эти два компонента; благодаря итераторам любой алгоритм может работать практически с любым контейнером; универсальность алгоритмов ещё более увеличивается с приминением функторов (функциональных объектов), реализованных в STL

- библиотека STL состоит из 5 основных компонентов:

1. Контейнер ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *container*) — хранение набора объектов в памяти.
2. Итератор ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *iterator*) — обеспечение средств доступа к содержимому контейнера.
3. Алгоритм ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *algorithm*) — определение вычислительной процедуры.
4. Адаптер ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *adaptor*) — адаптация компонентов для обеспечения различного интерфейса.
5. Функциональный объект ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *functor*) — сокрытие функции в объекте для использования другими компонентами.

Такое разделение позволяет нам уменьшить количество компонентов. Например, вместо написания функции поиска элемента для каждого вида контейнера мы обеспечиваем единственную версию, которая работает с каждым из них, пока удовлетворяется основной набор требований.

    Следующее описание разъясняет структуру библиотеки. Если программные компоненты сведены в таблицу как трёхмерный массив, где одно измерение представляет различные типы данных (например, int, double), второе измерение представляет различные контейнеры (например, вектор, связный список, файл), а третье измерение представляет различные алгоритмы с контейнерами (например, поиск, сортировка, перемещение по кругу) , если i, j и k - размеры измерений, тогда должно быть разработано i\* j \*k различных версий кода. При использовании шаблонных функций, которые берут параметрами типы данных, нам нужно только j \* k версий. Далее, если заставим наши алгоритмы работать с различными контейнерами, то нам нужно просто j+k версий. Это значительно упрощает разработку программ, а также позволяет очень гибким способом использовать компоненты в библиотеке вместе с определяемыми пользователем компонентами. Пользователь может легко определить специализированный контейнерный класс и использовать для него библиотечную функцию сортировки. Для сортировки пользователь может выбрать какую-то другую функцию сравнения либо через обычный указатель на сравнивающую функцию, либо через функциональный объект (объект, для которого определён operator()), который сравнивает. Если пользователю необходимо выполнить передвижение через контейнер в обратном направлении, то используется адаптер reverse\_iterator.

### Контейнеры

- контейнеры STL реализуют часто востребованные структуры данных такие как динамический массив, стек, очередь и т.д.

- все контейнеры содрежат набор однотипных элементов

- Containers are objects that store other objects. They control allocation and deallocation of these objects through constructors, destructors, insert and erase operations. They are implemented as class templates, which allows a great flexibility in the types supported as elements.

- The container manages the storage space for its elements and provides member functions to access them, either directly or through iterators (reference objects with similar properties to pointers). For example, a list container includes functions that insert, delete, and merge elements. A stack includes functions that push and pop values.  
  
- Containers replicate structures very commonly used in programming: dynamic arrays ([vector](http://www.cplusplus.com/vector)), queues ([queue](http://www.cplusplus.com/queue)), stacks ([stack](http://www.cplusplus.com/stack)), heaps ([priority\_queue](http://www.cplusplus.com/priority_queue)), linked lists ([list](http://www.cplusplus.com/list)), trees ([set](http://www.cplusplus.com/set)), associative arrays ([map](http://www.cplusplus.com/map))...

- все контейнеры являются динамическими: один из параметров шаблона определяет по умолчанию распределитель памяти (allocator)

#### Контейнеры можно разделить на 4 категории:

1. Последовательные (sequence c-s):

- vector

- list

- deque

Последовательность (Sequence) – контейнер переменных размеров, элементы которого организованы в строгом линейном порядке; п. поддерживает вставку и удаление элементов

- последовательный контейнер – неотсортированная коллекция однотипных элементов, хранящихся в том порядке, в котором они были добавлены, т.е. позиция элемента не зависит от его значения

- The library provides three basic kinds of sequence containers: vector, list, and deque. It also provides container adaptors that make it easy to construct abstract data types, such as stacks orqueues, out of the basic sequence kinds (or out of other kinds of sequences that the user might define).

2. Ассоциативные:

- set

- multiset

- map

- multimap

Ассоциативные контейнеры представляют собой отсортированные по умолчанию коллекции однотипных элементов, в которых позиция элемента зависит от его значения.

Семантически а.к. являются реализацией двоичных деревьев поиска.

Associative containers provide an ability for fast retrieval of data based on keys.

An Associative Container is a variable-sized [Container](http://www.sgi.com/tech/stl/Container.html) that supports efficient retrieval of elements (values) based on keys. It supports insertion and removal of elements, but differs from a [Sequence](http://www.sgi.com/tech/stl/Sequence.html) in that it does not provide a mechanism for inserting an element at a specific position.

3. Контейнеры-адаптеры (container-adaptor):

- stack

- queue

- priority\_queue

[stack](http://www.cplusplus.com/stack), [queue](http://www.cplusplus.com/queue) and [priority\_queue](http://www.cplusplus.com/priority_queue) are implemented as *container adaptors*. Container adaptors are not full container classes, but classes that provide a specific interface relying on an object of one of the container classes (such as [deque](http://www.cplusplus.com/deque) or [list](http://www.cplusplus.com/list)) to handle the elements. The underlying container is encapsulated in such a way that its elements are accessed by the members of the *container class* independently of the underlying *container* class used.

4. Псевдоконтейнеры

- basic\_string

- bitset

- valarray

- одно из впечатляющих достоинств STL – возможность создавать собственные контейнеры; т.о. библиотеку можно расширять; как только вы описали к., он автоматически становится совместимым с остальным содержимым STL

- Many containers have several member functions in common, and share functionalities. The decision of which type of container to use for a specific need does not generally depend only on the functionality offered by the container, but also on the efficiency of some of its members (complexity). This is especially true for sequence containers, which offer different trade-offs in complexity between inserting/removing elements and accessing them.

- у разных контейнеров разная скорость выполнения базовых операций: добавление в начало/конец, вставка, удаление, произвольный доступ (по индексу)

- т.о. выбор контейнера определяется тем какие операции выполняются чаще в программе

#### Описание последовательных контейнеров

|  |  |  |  |
| --- | --- | --- | --- |
| Контейнер | Описание | Эффективные операции | Сложность |
| vector | динамический массив с автоматическим изменением размера при добавлении/удалении; vector занимает амортизированное  время | произвольный доступ | О(1) |
| добавление-удаление элемента в конец | О(1)\* |
| list | двусвязный список | вставка/удаление в середине | О(1) |
| deque | двусвязный список линейных массивов | вставка/удаление в начале/конце | О(1) |
| basic\_string | хранит последовательность символов; по характеристикам подобен вектору; поддерживает стандартные операции над строками, такие как поиск, копирование O(N) и конкатенация O(N); эффективный | произвольный доступ к элементам | О(1) |

- string – это typedef для basic\_string<char>

#### Описание ассоциативных контейнеров

|  |  |  |  |
| --- | --- | --- | --- |
| Контейнер | Описание | Эффективные операции | Сложность |
| set | Упорядоченное [множество](http://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE) уникальных элементов. При вставке/удалении элементов множества итераторы, указывающие на элементы этого множества, не становятся недействительными. Обеспечивает стандартные операции над множествами типа объединения, пересечения, вычитания. **Тип элементов множества должен реализовывать оператор сравнения operator< или требуется предоставить функцию-компаратор.** Реализован на основе самобалансирующего дерева двоичного поиска. |  |  |
|  |  |
| multiset |  |  |  |
| map | Упорядоченный [ассоциативный массив](http://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%81%D0%BE%D1%86%D0%B8%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2) пар элементов, состоящих из ключей и соответствующих им значений. Ключи должны быть уникальны. Порядок следования элементов определяется ключами. **При этом тип ключа должен реализовывать оператор сравнения operator<, либо требуется предоставить функцию-компаратор.** |  |  |
| multimap |  |  |  |

- priority\_queue – модель кучи

#### Требования к классам объектов, передаваемых в контейнеры в качестве элементов:

В контейнерах для хранения элементов используется семантика передачи объектов по значению. Другими словами, при добавлении контейнер получает копию элемента. Если создание копии нежелательно, то используют контейнер указателей на элементы. Присвоение элементов реализуется с помощью оператора присваивания, а их разрушение происходит с использованием деструктора.

* Требуется возможность копирования элемента копирующим конструктором. Созданная копия должна быть эквивалентна оригиналу. Это означает, что любая проверка на равенство должна считать копию и оригинал равными, а поведение копии не должно отличаться от поведения оригинала.

    Все контейнеры создают внутренние копии своих элементов и возвращают их временные копии. Копирующий конструктор вызывается очень часто, поэтому он должен обладать хорошим быстродействием (это не требование, а всего лишь условие, повышающее эффективность работы контейнера). Если копирование объектов занимает слишком много времени, попробуйте предотвратить копирование, используя контейнер со ссылочной семантикой.

* Требуется возможность присваивания элемента оператором присваивания. Контейнеры и алгоритмы используют оператор присваивания для замены старых элементов новыми.
* Требуется возможность уничтожения элемента деструктором. Контейнеры уничтожают свои внутренние копии элементов при удалении этих элементов из контейнера. Следовательно, деструктор не должен быть закрытым (**private**). Кроме того, как обычно в **C++**, деструктор не должен инициировать исключения, иначе возможны абсолютно непредсказуемые последствия. Эти три операции - копирование, присваивание, уничтожение - автоматически генерируются для любого класса. Следовательно, любой класс автоматически удовлетворяет требованиям, если для него не были определены специальные версии этих операций или их нормальная работа не нарушается другими членами класса.

    Кроме того, к элементам могут предъявляться дополнительные требования.

* Для некоторых функций классов последовательных контейнеров требуется конструктор по умолчанию. Например, можно создать непустой контейнер или увеличить количество элементов без указания значений новых элементов. Такие элементы создаются вызовом конструктора по умолчанию для соответствующего типа.
* Для некоторых операций требуется определить проверку на равенство оператором ==. Такая необходимость особенно часто возникает при поиске.
* Для ассоциативных контейнеров требуется, чтобы элементы поддерживали критерий сортировки. По умолчанию используется оператор <, вызываемый объектом функции **less<>**.

**Основные требования к элементам в контейнерах**

|  |  |  |
| --- | --- | --- |
| **Метод** | **Описание** | **Примечание** |
| Конструктор копии | Создает новый элемент, идентичный старому | Используется при каждой вставке элемента в контейнер |
| Оператор присваивания | Заменяет содержимое элемента копией исходного элемента | Используется при каждой модификации элемента |
| Деструктор | Разрушает элемент | Используется при каждом удалении элемента |
| Конструктор по умолчанию | Создает элемент без аргументов | Применяется только для определенных операций |
| operator== | Сравнивает два элемента | Используется при выполнении operator== для двух контейнеров |
| operator< | Определяет, меньше ли один элемент другого | Используется при выполнении operator< для двух контейнеров |

- требования для классов объектов, передаваемых в последовательные контейнеры:

- конструктор копий

- оператор присваивания

- требования для классов объектов, передаваемых в ассоциативные контейнеры:

- конструктор копий

- оператор присваивания

- после вставки итератор будет указывать на вставленный элемент

## Многозадачность

МНОГОПРОЦЕССНОСТЬ

(процессная многозадачность)

МНОГОПОТОЧНОСТЬ

(поточная многозадчность)

Многозадачность – свойство среды выполнения обезпечивать выполнение нескольких задач параллельно.

Многозадачность может быть реализована на много- процессорной/ядерной системе, за счёт использования программой:

1) Нескольких процессов

2) Нескольких потоков

- многопоточность является *частным случаем многозадачности*, т.к. последняя может быть процессной и поточной

- поток – путь исполнения внутри процесса (при запуске программы ОС создаёт её процесс и начинает выполнять основной поток этого процесса; основной поток передаётся ОС в виде адреса функции, обычно это функция main или WinMain)

- потоки можно создавать внутри процесса (All threads in MFC applications are represented by [CWinThread](http://msdn.microsoft.com/en-us/library/48xz4yz9.aspx) objects)

- In most situations, you do not even have to explicitly create these objects; instead call the framework helper function [AfxBeginThread](http://msdn.microsoft.com/en-us/library/s3w9x78e.aspx), which creates the **CWinThread** object for you.

- в MFC различают два вида потоков: 1) потоки пользовательского интерфейса и 2) рабочие потоки (worker threads)

- потоки пользовательского интерфейса служат для обработки пользовательского ввода, а также – обслуживания событий и сообщений, генерируемых пользователем

- рабочие потоки (worker threads) нужны для выполнения заданий, не требующих пользовательского ввода (например выполнение каких-либо вычислений)

- Win32 API не различает типы потоков, ему нужно знать только начальный адрес потока для того, чтобы начать выполнять его

- основной трудностью при написании многопоточных приложений является разрешение ситуаций, когда более чем один поток нуждается в доступе к одному и тому же объекту (см. [Multithreading: Programming Tips](http://msdn.microsoft.com/en-us/library/h14y172e.aspx))

- you must ensure that objects are not accessed by more than one thread at a time

--------------------------------

#### Процесс

[Процесс (информатика)](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) — выполнение инструкций компьютерной программы на процессоре ЭВМ.

Процесс состоит из команд программы и имеет своё адресное пространство – это значит, что команды программы выполняются в отдельной области памяти. Отсюда следует, что разные процессы могут иметь одинаковые наборы относительных адресов памяти. Т.е. данные процесса А, размещённые по адресу 0xffade45 и данные процесса Б, размещённые по тому же адресу 0xffade45, будут существовать независимо друг от друга.

У процесса есть свои:

1) Адресное пространство

2) Программный счётчик (program counter) – помнит какое выражение должно быть выполнено следующим

3) Стек – содержит аргументы, переданные функциям и локальные переменные функций

4) Куча – для хранения данных, которые должны существовать дольше выполнения функций (malloc, new)

- процесс – 1) совокупность команд, представляющих программу, выполняющихся в адресном пространстве; 2) команда, которая выполняется в текущий момент; 3) совокупность взаимосвязанных и взаимодействующих действий, преобразующих входящие данные в исходящие (ISO 9000:2000, definitions); 3) компьютерная программа сама по себе это только пассивная совокупность инструкций, в то время как процесс — это непосредственное выполнение этих инструкций; 4) часто процессом называют выполняющуюся программу и все её элементы: адресное пространство, глобальные переменные, регистры, стек, открытые файлы и т. д.; 5) A process is an executing instance of an application (MSDN, Multithreading with C++ and MFC); 6) An application consists of one or more processes. A process, in the simplest terms, is an executing program (MSDN, Processes and Threads).

- процесс может порождать процесс

- во время запуска программы для неё выделяется набор ячеек виртуальной памяти, называемый адресным пространством

#### Поток

[Поток выполнения](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F), нить ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *thread*) — в программировании наименьшая единица обработки, которая может быть назначена операционной системой на исполнение процессору. В большинстве случаев поток находится внутри процесса.

Потоки порождаются процессом и существуют внутри него.

Потоки могут порождать потоки, однако все потоки принадлежат процессу. Поток не может принадлежать другому потоку.

Во время создания процесса ему автоматически выделяется поток, называющийся «главный поток» - это обычный поток как и любой другой; название связано с тем, что он создаётся первым.

Когда главный поток процесса уничтожается, процесс также уничтожается (native C++).

Когда последний поток процесса уничтожается, процесс также уничтожается (C++/CLI).

В отличие от процесса поток не имеет отдельного адресного пространства. Все потоки процесса разделяют между собой его адресное пространство и ресурсы. Когда программа состоит из двух или более потоков все потоки имеют (разделяют) одно адресное пространство. И если поток А изменит данные по адресу 0xffade45, то поток Б и остальные потоки сразу же увидят эти изменения.

Все потоки имеют (разделяют) одну и ту же кучу. Если, например, один поток зарезервирует для себя всю кучу, то все остальные потоки уже не смогут выделить себе динамическую память.

Каждый поток имеет свой отдельный стек. Это значит, что поток А может выполнять функцию Ф1, параллельно с выполнением потоком Б функции Ф2.

Каждый поток имеет свои отдельные регистры.

Потоки не имеют своих отдельных и *разделяют* *принадлежащие процессу*:

1) Адресное пространство

2) Кучу

Потоки имеют свои отдельные:

1) Стек

2) Регистры

Выполнение потока прерывается:

1) Когда потоку нужно получить новые ресурсы

2) Когда потоку надо подождать завершения какой-либо операции (такой как I/O)

3) Когда поток выработал отведённое ему процессорное время

Когда процессор переходит от выполнения одного потока к выполнению другого потока, это называется переключением контекста (context switching).

### Многопоточность

Работа, выполняемая процессом, может быть разделена между различными потоками. Это называется многопоточностью (multithreading).

Если у тебя нет разделяемых данных, то никакой синхронизации потоков не должно быть. Они просто выполняются паралелльно.

#### Создание потоков

Для создания потока на Win32 API:

uintptr\_t \_beginthread(

void(\_\_cdecl \*start\_address)(void \*),

unsigned stack\_size,

void \*arglist

);

start\_address – указатель на функцию, с которой начнётся выполнение потока, entry-point-function

stack\_size – размер стека для данного потока, 0 – для автоматического определения

arglist – любой параметр

Пример:

#include <iostream>

#include <Windows.h>

#include <process.h>

using namespace std;

void thread\_entry\_point\_func(void\* p)

{

cout << "Entered new thread " << (int\*)p << "\n\n";

}

void main()

{

\_beginthread(thread\_entry\_point\_func, 0, (void\*)1024);

Sleep(100);

}

VOID WINAPI Sleep(\_In\_ DWORD dwMilliseconds) – приостанавливает выполнение потока, в котором она вызвана, на указанное количество миллисекунд; в это время будут выполняться другие имеющиеся потоки; значение аргумента равное 0 заставляет поток передать остаток своего кванта времени любому другому потоку готовому к выполнению; если же других потоков нет, то функция тут же возвращается и поток продолжает своё выполнение.

В данном случае будет создан главный поток приложения с функцией входа main(), который выполнит функцию main() до создания второго потока в \_beginthread и будет приостановлен на 100 миллисекунд. В этом промежутке времени будет создан (успеет создаться) второй поток с функцией входа thread\_entry\_point\_func, которая выведет своё сообщение в консоль. После чего будет уничтожен главный поток main и вместе с ним будут уничтожены все остальные потоки и сам процесс.

Если закоментировать вызов Sleep(100), то главный поток дойдёт до конца функции main и завершится вместе с процессом. Второй поток не успеет создаться и следовательно его сообщение не будет выведено в консоль.

# DirectX

### VC++ Directories

$(DXSDK\_DIR)Include;$(DXSDK\_DIR)Samples\C++\DXUT\Core;$(DXSDK\_DIR)Samples\C++\DXUT\Optional;

$(NVSDK10D3D\_ROOT)\Include\DXUT;$(NVSDK10D3D\_ROOT)\Include\FCollada;$(NVSDK10D3D\_ROOT)\Include\NVUT;$(NVSDK10D3D\_ROOT)\Include\zlib;

$(DXSDK\_DIR)Lib\x86;$(DXSDK\_DIR)Lib\x64;

$(NVSDK10D3D\_ROOT)\Lib

C:\Program Files (x86)\Microsoft DirectX SDK (June 2010)\Include

C:\ProgramData\NVIDIA Corporation\NVIDIA Direct3D SDK 10\Include\DXUT

C:\ProgramData\NVIDIA Corporation\NVIDIA Direct3D SDK 10\Include\FCollada

C:\ProgramData\NVIDIA Corporation\NVIDIA Direct3D SDK 10\Include\NVUT

C:\ProgramData\NVIDIA Corporation\NVIDIA Direct3D SDK 10\Include\zlib

C:\ProgramData\NVIDIA Corporation\NVIDIA Direct3D SDK 10\Lib

C:\Program Files (x86)\Microsoft DirectX SDK (June 2010)\Lib\x64

C:\Program Files (x86)\Microsoft DirectX SDK (June 2010)\Lib\x86

## Библиотека DirectXMath

The DirectXMath Library implements an optimal and portable interface for arithmetic and linear algebra operations on single-precision floating-point vectors (2D, 3D, and 4D) or matrices (3×3 and 4×4). The DirectXMath library is based on the [XNA Math C++ SIMD library version 2.04](http://blogs.msdn.com/b/chuckw/archive/2011/02/23/xna-math-version-2-04.aspx). The library has some limited support for integer vector operations. These operations are used extensively in rendering and animation by graphics programs. There is no support for double-precision vectors (including longs, shorts, or bytes), and only limited integer vector operations.

The library is available on a variety of Windows platforms. Because the library provides functionality not available previously, this version supersedes the following libraries:

* Xbox Math library provided by the Xboxmath.h header
* D3DX 9 library provided by the D3DX 9 DLLs
* D3DX 10 math library provided through the D3DX 10 DLLs
* XNA Math library provided by the xnamath.h header in the DirectX SDK and Xbox 360 XDK

The DirectXMath Library primarily supports the C++ programming language. The library is implemented using inline routines in the header files, DirectXMath\*.inl, DirectXPackedVector.inl and DirectXCollision.inl. This implementation makes use of high-performance compiler intrinsics.

The DirectXMath Library provides:

* An implementation using SSE/SSE2 intrinsics.
* An implementation without intrinsics.
* An implementation using ARM-NEON intrinsics.

DirectXMath differences from XNA Math

Here is how the DirectXMath library primarily differs from the XNA Math library:

* DirectXMath is C++ only (namespaces, overloads, new templates, and so on).
* Requires C++11 standard library support (that is, stdint.h, and so on).
* ARM-NEON intrinsics support for the Windows RT platform.
* New color functionality (color space conversions, .NET color constants).
* Bounding volume types (a version of which was previously in the XNACollision header in the DirectX SDK Collision sample).
* No Xbox 360 version is available. The Xbox 360 XDK continues to ship XNAMath v2.x; removal of Xbox 360 specific data types and function variants.
* Reworked [**XMVectorPermute**](http://msdn.microsoft.com/en-us/library/windows/desktop/hh855956(v=vs.85).aspx) for improved optimization for SSE and ARM-NEON intrinsics.
* The [**XMMATRIX**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmmatrix(v=vs.85).aspx) type is fully opaque. To access individual elements of **XMMATRIX**, use other types such as[**XMFLOAT4X4**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat4x4(v=vs.85).aspx).

### Подключение библиотеки

- **To use DirectXMath Library functions, include the DirectXMath.h, DirectXPackedVector.h, DirectXColors.h, and/or DirectXCollision.h headers. The headers are found in the Windows Software Development Kit for Windows Store apps.**

- **The DirectXMath library uses C++ namespaces to organize the types. XNA Math used only the global namespace.** The DirectXMath types in common with XNA Math are in the “DirectX” or the “DirectX::PackedVector” namespace.

- **In C++ source files, a simple solution is to add ‘using’ statements:**

#include “DirectXMath.h”

#include “DirectXPackedVector.h”

using namespace DirectX;

using namespace DirectX::PackedVector;

**For headers, it is not considered ‘best practice’ to add using statements. Instead, add fully qualified namespaces:**

struct mystruct

{

DirectX::XMFLOAT3 position;

DirectX::PackedVector::HALF packedValue;

};

- For various functions that load less than 4 elements of an XMVECTOR, the XNA Math library left the additional elements undefined. DirectXMath will always fill these additional elements with 0.

-D3DXMath is a math helper library for Direct3D applications. D3DXMath is long-standing, is included in D3DX 9 and D3DX 10, and dates back to older versions of DirectX as well.

- **The D3DX utility library (D3DX 9, D3DX 10, and D3DX 11) is deprecated for Windows 8, so we highly recommended that you migrate to DirectXMath rather than using D3DXMath.**

- DirectXMath shares much of the same functionality in D3DXMath, and internally D3DXMath includes a number of processor-specific optimizations. **The key difference is that D3DXMath is hosted in the D3DX9\*.DLLs and D3DX10\*.DLLs, and very few of the functions are inlined. The DirectXMath Library calling convention is explicitly SIMD friendly, whereas D3DXMath has to perform load and store conversions to implement SIMD optimization.**

- **To write code that gets and sets individual components of a vector in a portable and optimal manner use DirectXMath vector accessor functions.**

### Основные типы данных

The [**XMVECTOR**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee420742(v=vs.85).aspx) and [**XMMATRIX**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmmatrix(v=vs.85).aspx) types are the work horses for the DirectXMath Library. Every operation consumes or produces data of these types. Working with them is key to using the library. However, since DirectXMath makes use of the SIMD instruction sets, these data types are subject to a number of restrictions. It is critical that you understand these restrictions if you want to make good use of the DirectXMath functions.

#### Способы обращения с данными типов XMVECTOR и XMVECTOR

- The most efficient form of vector processing is to use memory-to-memory streaming where the input data is loaded from memory (using Vector Load Functions), processed fully in SIMD form, and then written to memory (using Vector Store Functions).

**- Developers should use DirectXMath Library's**[**accessor**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee415582(v=vs.85).aspx)**,**[**load**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee415589(v=vs.85).aspx)**, and**[**store**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee415635(v=vs.85).aspx)**functions to get and set the vectors, and the**[**DirectXMath Library 4D Vector Functions**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee415649(v=vs.85).aspx)**to manipulate them.**

**For projects that need detailed information about how to implement XMVECTOR on different platforms, see**[**Library Internals**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee418728(v=vs.85).aspx)**.**

#### Правила употребления типов

- You should think of [**XMVECTOR**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee420742(v=vs.85).aspx) as a proxy for a SIMD hardware register, and [**XMMATRIX**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmmatrix(v=vs.85).aspx) as a proxy for a logical grouping of four SIMD hardware registers.

- SIMD инструкции на версиях Windows, поддерживающих SSE2, имеют aligned и unaligned версии операций над памятью. Использование aligned операций быстрее и должно использоваться везде, где это возможно

- Библиотека DirectXMath предоставляет доступ к aligned функциональности посредством соответствующих вариантов векторных типов, структур и функций, имена которых заканчиваются на A

- For example, there are an unaligned [**XMFLOAT4X4**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat4x4(v=vs.85).aspx) structure and an aligned [**XMFLOAT4X4A**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat4x4a(v=vs.85).aspx) structure, which is used by the [**XMStoreFloat4**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.storing.xmstorefloat4(v=vs.85).aspx) and [**XMStoreFloat4A**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.storing.xmstorefloat4a(v=vs.85).aspx) functions respectively

- В основе DirectXMath лежат aligned SSE intrinsics. По этой причине операции, использующие XMVECTOR и XMMATRIX, подразумевают что объекты этих типов имеют 16-байтное выравнивание

- **Для стековых выделений** обезпечивается **автоматическое 16-байтное** выравнивание для XMVECTOR и XMMATRIX (if code is compiled against the DirectXMath Library using the recommended Windows (см. «Правильные настройки компиляции») compiler settings)

- Однако, **для выделений в куче** (**XMVECTOR\*** и **XMMATRIX\*** ) **автоматического** выравнивания **не происходит** (так, например, для платформы Windows x64 все выделения в куче имеют 16-байтное выравнивание, а для Windows x86 – 8-байтное). Этого надо добиваться дополнительными средствами (например, [\_aligned\_malloc](http://msdn.microsoft.com/en-us/library/8z34s9c6(VS.80).aspx) – для контроля за выравниванием памяти при выделении в куче). Т.о. при выделении динамической памяти или во время преобразования к типам XMVECTOR \* и XMMATRIX\* нужно удостовериться в соблюдении выравнивания по 16-байтной границе

- Используя **aligned типы** (напр. XMVECTOR и XMMATRIX) DirectXMath **с контейнерами** библиотеки **STL**, **нужно** предоставить **пользовательский** **распределитель** (**allocator), который гарантирует 16-байтное** выравнивание. See the Visual C++ Team [blog](http://blogs.msdn.com/b/vcblog/archive/2008/08/28/the-mallocator.aspx) for an example of writing a custom allocator (вместо malloc/free **нужно будет использовать \_aligned\_malloc/\_aligned\_free**)

##### **Правильные настрйоки компиляции:**

For Windows x86 targets, enable **/arch:SSE2**. For all Windows targets, enable /fp:fast.

By default, compilation against the DirectXMath Library for Window x86 targets is done with \_XM\_SSE\_INTRINSICS\_ defined. This means that all DirectXMath functionality will make use of SSE2 instructions. However, the same is not true for other code.

Code outside of DirectXMath is handled using compiler defaults. Without this switch, the generated code may often use the less efficient x87 code.

##### **Итого:**

- Рекомендуется всегда использовать XMVECTOR и XMMATRIX из-за совместимости таких данных с SIMD регистрами, т.к. XMVECTOR и XMMATRIX являются их программными соответствиями

- **Если можно гарантировать выравнивание данных по 16-байтной границе, то используются XMVECTOR и XMMATRIX, иначе используются** [**XMFLOAT3**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat3(v=vs.85).aspx), [**XMFLOAT4**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat4(v=vs.85).aspx), [**XMFLOAT4X3**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat4x3(v=vs.85).aspx), [**XMFLOAT4X4**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat4x4(v=vs.85).aspx) **(что не желательно)**

**- в случаях, когда автоматическое выранивание не обезпечивается для типов XMVECTOR и XMMATRIX, следует обезпечить его вручную нежели использовать unaligned аналоги этих типов (например, перегрузить глобальную функцию operator new() )**

**- обезпечить правильные настройки компиляции:** For Windows x86 targets, enable /arch:SSE2. For all Windows targets, enable /fp:fast

- **при выделениях в стеке и сегментах:** для объявления глобальных и локальных переменных используются XMVECTOR и XMMATRIX (в этом случае компилятор сам обезпечит правильное размещение данных в памяти - the compiler will automatically place them correctly on the stack when they are used as a local variable, or place them in the data segment when they are used as a global variable)

- **при выделениях в куче:** при необходимости создания полей класса в виде указателей на типы XMVECTOR и XMMATRIX для простоты можно использовать аналоги этих типов - [**XMFLOAT3**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat3(v=vs.85).aspx), [**XMFLOAT4**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat4(v=vs.85).aspx), [**XMFLOAT4X3**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat4x3(v=vs.85).aspx), [**XMFLOAT4X4**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat4x4(v=vs.85).aspx), при этом операции над данными будут не эффективными из-за отсутствия выравнивания. Связано это с рядом дополнительных требований по обезпечению 16-байтовой границы, которые надо выполнить при выделениях в динамической памяти

- **при передаче параметров в функции:** можно использовать XMVECTOR и XMMATRIX, **используя алиасы**, которые будут указывать компилятору соответствующее **соглашение вызова в зависимости от платформы** (with proper conventions, they can also be passed safely as parameters to a function (see [Calling Conventions](http://msdn.microsoft.com/en-us/library/windows/desktop/ee418728(v=vs.85).aspx#Call_Conventions) for details))

- **при работе с STL:** нужно написать пользовательский распределитель (allocator), реализующий выделение в куче с 16-байтным выравниванием

#### XMVECTOR

- XMVECTOR Data Type - a portable type used to represent a vector of four 32-bit floating-point or integer components, each aligned optimally and mapped to a hardware vector register

In the DirectXMath Library, to fully support portability and optimization, XMVECTOR is, by design, **an opaque type**. The actual implementation of XMVECTOR is platform dependent.

In general, code should not rely on the specifics of any given platform specific implementation of XMVECTOR.

For a list of additional functionality, such as constructors and operators, available using XMVECTOR when programming in C++, see [XMVECTOR Extensions](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.types.xmvector_members(v=vs.85).aspx):

|  |  |
| --- | --- |
| **Methods** | **Description** |
| [**operator +=**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee421394(v=vs.85).aspx) | Adds a floating point value to an XMVECTOR instance, and returns a reference to the updated instance. |
| [**operator -=**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee421383(v=vs.85).aspx) | Subtracts a floating point value from the current instance of XMVECTOR, returning the result in the updated current instance. |
| [**operator \***](http://msdn.microsoft.com/en-us/library/windows/desktop/ee421393(v=vs.85).aspx) | Multiplication operators |
| [**operator \*=**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee421389(v=vs.85).aspx) | Multiplication assignment operators |
| [**operator /**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee421382(v=vs.85).aspx) | Division operator. |
| [**operator /=**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee421379(v=vs.85).aspx) | Division Assignment operator. |
| [**operator -**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee421386(v=vs.85).aspx) | Subtraction and negation operators |
| [**operator +**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee421397(v=vs.85).aspx) | Addition operators. |

For XNAMATH 2.x, the XMVECTOR data type has .x, .y, .z, .and .w element members, which generally cause poor performance. The use of the XM\_STRICT\_VECTOR4 type provides an opt-in of the DirectXMath definition of an opaque data type.

**Header:** DirectXMath.h

**Namespace**: Use DirectX

Platform Requirements

Microsoft Visual Studio 2010 or Microsoft Visual Studio 2012 with the Windows SDK for Windows 8. Supported for **Win32 desktop apps**, Windows Store apps, and Windows Phone 8 apps.

#### XMMATRIX

struct XMMATRIX {

XMVECTOR r[4]; // Array of four vectors, representing the rows of the matrix.

};

- **XMMATRIX** structure - describes a 4\*4 matrix aligned on a 16-byte boundary that maps to four hardware vector registers.

For a list of additional functionality such as constructors and operators that are available using XMMATRIX when you are programming in C++, see [XMMATRIX Extensions](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmmatrix_members(v=vs.85).aspx):

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| [**XMMATRIX (float,float,float,float,float,float,float,float,float,float,float,float,float,float,float,float)**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee420126(v=vs.85).aspx) | Initializes a new instance of the XMMATRIX structure from sixteen scalar float values.  Initializes a new instance of the[**XMMATRIX**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmmatrix(v=vs.85).aspx) structure from sixteen scalar float values.  **Note**   This constructor is only available when developing with C++. |
| [**XMMATRIX ()**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee420023(v=vs.85).aspx) | Default constructor forXMMATRIX.  Default constructor for[**XMMATRIX**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmmatrix(v=vs.85).aspx).  **Note**  This constructor is only available when developing with C++. |
| [**XMMATRIX (const float\*)**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee420024(v=vs.85).aspx) | Initializes a new instance of theXMMATRIX structure from a sixteen element float array.  Initializes a new instance of the[**XMMATRIX**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmmatrix(v=vs.85).aspx) structure from a sixteen element float array.  **Note**  This constructor is only available when developing with C++. |
| [**XMMATRIX (XMVECTOR,XMVECTOR,XMVECTOR,XMVECTOR)**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee420127(v=vs.85).aspx) | Initializes a new instance of theXMMATRIX structure from four instances of XMVECTOR.  Initializes a new instance of the[**XMMATRIX**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmmatrix(v=vs.85).aspx) structure from four instances of [**XMVECTOR Data Type**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee420742(v=vs.85).aspx) |

|  |  |
| --- | --- |
| **Methods** | **Description** |
| [**operator \***](http://msdn.microsoft.com/en-us/library/windows/desktop/ff621686(v=vs.85).aspx) | Performs a matrix multiplication of the current instance of XMMATRIX by another instance of XMMATRIX. |
| [**operator \*=**](http://msdn.microsoft.com/en-us/library/windows/desktop/ff621687(v=vs.85).aspx) | Performs a matrix multiplication of the current instance of XMMATRIX by another instance of XMMATRIX and returns a reference to the current instance, which has been updated. |
| [**operator =**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee420128(v=vs.85).aspx) | Assigns the matrix data of one instance of XMMATRIX to the current instance of XMMATRIX and returns areference to the current instance. |
| [**operator ()**](http://msdn.microsoft.com/en-us/library/windows/desktop/ff621685(v=vs.85).aspx) | Accesses specific matrix elements referenced by row and column from the current instance of XMMATRIX. |

In the DirectXMath.h header file, the system uses an alias to the XMMATRIX object, specifically **CXMMATRIX**. The header uses the alias to comply with the optimal in-line calling conventions of different compilers. For most projects using DirectXMath it is sufficient to simply treat this as an exact alias to XMMATRIX.

typedef const XMMATRIX CXMMATRIX;

For projects that need detailed information about how different platform's calling conventions are handled, see [Library Internals](http://msdn.microsoft.com/en-us/library/windows/desktop/ee418728(v=vs.85).aspx).

**XMMATRIX is row-major and all DirectXMath functions that accept an XMMATRIX as a parameter expect data to be organized as row-major.**

Data in an XMMATRIX has the following layout.

\_11 \_12 \_13 \_14

\_21 \_22 \_23 \_24

\_31 \_32 \_33 \_34

\_41 \_42 \_43 \_44

DirectXMath defines **XMMATRIX** as a **fully opaque type**. To access individual elements of **XMMATRIX**, use equivalent types such as [**XMFLOAT4**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat4(v=vs.85).aspx) for a given row or [**XMFLOAT4X4**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat4x4(v=vs.85).aspx) for the whole matrix.

**Note**  XNAMath 2.x defines XMMATRIX as a union with **\_11** to **\_44** members and an **m** array member. When you use these members of the union, poor performance results. DirectXMath.h still defines these XMMATRIX union members for when you build an app with [\_XM\_NO\_INTRINSICS\_](http://msdn.microsoft.com/en-us/library/windows/desktop/ee415579(v=vs.85).aspx). XNAMath version 2.05 provides an opt-in XM\_STRICT\_XMMATRIX to enforce the DirectXMath behavior.

**Namespace:** Use DirectX

**- Compiler Aliases**

The DirectXMath.h header file uses aliases to the XMVECTOR object, specifically **CXMVECTOR** and **FXMVECTOR**. **The header uses these aliases to comply with the optimal in-line calling conventions of different compilers.** **For most projects that use DirectXMath it is sufficient to treat these types as an exact alias to XMVECTOR.**

For example:

[CDATA[

typedef const XMVECTOR FXMVECTOR;

typedef const XMVECTOR CXMVECTOR;

]]

For projects that need detailed information about how different platforms handle their calling conventions, see [Library Internals](http://msdn.microsoft.com/en-us/library/windows/desktop/ee418728(v=vs.85).aspx).

**Header:** DirectXMath.h

**Namespace:** Use DirectX

Platform Requirements

Microsoft Visual Studio 2010 or Microsoft Visual Studio 2012 with the Windows SDK for Windows 8. Supported for **Win32 desktop apps**, Windows Store apps, and Windows Phone 8 apps.

#### Обезпечение 16-байтового выравнивания вручную

- **For projects that need detailed information about how to implement XMVECTOR on different platforms, see**[**Library Internals**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee418728(v=vs.85).aspx)**.**

- On Windows x64, all heap allocations are 16-byte aligned, but for Windows x86, they are only 8-byte aligned. There are options for allocating structures from the heap with 16-byte alignment (see [Properly Align Allocations](http://msdn.microsoft.com/en-us/library/windows/desktop/ee418732(v=vs.85).aspx#Properly_Align_Alloc)). For C++ programs, you can use operator new/delete/new[]/delete[] overloads (either globally or class-specific) to enforce optimal alignment if desired.

**Note**  As an alternative to enforcing alignment in your C++ class directly by overloading new/delete, you can use the [pImpl idiom](http://en.wikipedia.org/wiki/Opaque_pointer). If you ensure your **Impl** class is aligned via [**\_\_aligned\_malloc**](http://msdn.microsoft.com/en-us/library/windows/desktop/8z34s9c6(v=vs.85).aspx) internally, you can then freely use aligned types within the internal implementation. This is a good option when the 'public' class is a Windows Runtime ref class or intended for use with [**std::shared\_ptr<>**](http://msdn.microsoft.com/en-us/library/windows/desktop/bb982026(v=vs.85).aspx), which can otherwise disrupt careful alignment.

- **STL allocator**

### Наиболее часто употребляемые функции

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Алгоритм |
| XMVector3Dot | скалярно умножает два вектора XMVECTOR, как 3D-векторы; возврат: XMVECTOR с репликами скалярного произведения |  |
| XMVector3Cross | векторно умножает два вектора XMVECTOR, как 3D-векторы; возврат: XMVECTOR |  |
| XMVector3Transform | афинное преобразование вектора XMVECTOR матрицей XMMATRIX, w компонента вектора подразумевается равной 1 | XMVECTOR Z = XMVectorSplatZ(V);  XMVECTOR Y = XMVectorSplatY(V);  XMVECTOR X = XMVectorSplatX(V);  XMVECTOR Result = XMVectorMultiplyAdd(Z, M.r[2], M.r[3]);  Result = XMVectorMultiplyAdd(Y, M.r[1], Result);  Result = XMVectorMultiplyAdd(X, M.r[0], Result);  return Result; |
| XMVector3TransformNormal | только вращательное преобразование вектора XMVECTOR матрицей XMMATRIX, компонента w не учитывается | XMVECTOR Z = XMVectorSplatZ(V);  XMVECTOR Y = XMVectorSplatY(V);  XMVECTOR X = XMVectorSplatX(V);  XMVECTOR Result = XMVectorMultiply(Z, M.r[2]);  Result = XMVectorMultiplyAdd(Y, M.r[1], Result);  Result = XMVectorMultiplyAdd(X, M.r[0], Result);  return Result; |
| XMMatrixLookAtLH | возвращает матрицу перехода к базису, составленному из переданных параметров (векторы полученного базиса будут записаны в матрице по столбцам, т.е. это соответствует требованию к матрице перехода из глобального протсранства к объектному пространству); => можно использовать когда нужна матрица перехода к ВП (напр. камеры или ИС) |  |

### Другие важные функции

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Алгоритм |
| XMVectorSplat[X/Y/Z/W](XMVECTOR V) | Returns a vector, all of whose components are equal to the x/y/z or w component of V |  |
| XMVectorMultiplyAdd(v1, v2, v3) | Computes the product of the first two vectors added to the third vector | XMVECTOR Result;  Result.x = V1.x \* V2.x + V3.x;  Result.y = V1.y \* V2.y+ V3.y;  Result.z = V1.z \* V2.z+ V3.z;  Result.w = V1.w \* V2.w+ V3.w;  return Result; |
| XMVector3TransformCoord | то же что и XMVector3Transform, только результирующий вектор делится на свою же w-координату | XMVECTOR Z = XMVectorSplatZ(V);  XMVECTOR Y = XMVectorSplatY(V);  XMVECTOR X = XMVectorSplatX(V);  XMVECTOR Result = XMVectorMultiplyAdd(Z, M.r[2], M.r[3]);  Result = XMVectorMultiplyAdd(Y, M.r[1], Result);  Result = XMVectorMultiplyAdd(X, M.r[0], Result);  XMVECTOR W = XMVectorSplatW(Result);  return XMVectorDivide( Result, W ); |
| XMVectorSet(float x, float y, float z, float w) | функция для инициализации и установки значения векторов XMVECTOR; создаёт и возвращает вектор XMVECTOR; предоставляет единственный способ инициализации/присвоения заданного значения вектору (использовать вместо конструктора) |  |
| XMVectorZero() | возвращает нулевой вектор | XMVECTOR vResult = {0.0f,0.0f,0.0f,0.0f};  return vResult; |

### Соответствия между D3DXMath и DirectXMath

|  |  |
| --- | --- |
| **D3DXMath Type** | **DirectXMath Equivalent** |
| D3DXFLOAT16 | [**HALF**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.types.__macro_half(v=vs.85).aspx) |
| D3DXMATRIXA16 | [**XMMATRIX**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmmatrix(v=vs.85).aspx) or [**XMFLOAT4X4A**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat4x4a(v=vs.85).aspx) |
| D3DXQUATERNION  D3DXPLANE  D3DXCOLOR | [**XMVECTOR**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee420742(v=vs.85).aspx) is used rather than having unique types, so you will likely need to use an [**XMFLOAT4**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat4(v=vs.85).aspx) |
| D3DXVECTOR2 | [**XMFLOAT2**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat2(v=vs.85).aspx) |
| D3DXVECTOR2\_16F | [**XMHALF2**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmhalf2(v=vs.85).aspx) |
| D3DXVECTOR3 | [**XMFLOAT3**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat3(v=vs.85).aspx) |
| D3DXVECTOR4 | [**XMFLOAT4**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat4(v=vs.85).aspx)  (or if you can guarantee the data is 16-byte aligned, [**XMVECTOR**](http://msdn.microsoft.com/en-us/library/windows/desktop/ee420742(v=vs.85).aspx) or [**XMFLOAT4A**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmfloat4a(v=vs.85).aspx) ) |
| D3DXVECTOR4\_16F | [**XMHALF4**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.reference.xmhalf4(v=vs.85).aspx) |

|  |  |
| --- | --- |
| **D3DXMath Macro** | **DirectXMath Equivalent** |
| D3DX\_PI | [XM\_PI](http://msdn.microsoft.com/en-us/library/windows/desktop/ee415576(v=vs.85).aspx) |
| D3DX\_1BYPI | [XM\_1DIVPI](http://msdn.microsoft.com/en-us/library/windows/desktop/ee415576(v=vs.85).aspx) |
| D3DXToRadian | [**XMConvertToRadians**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.conversion.xmconverttoradians(v=vs.85).aspx) |
| D3DXToDegree | [**XMConvertToDegrees**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.conversion.xmconverttodegrees(v=vs.85).aspx) |

|  |  |
| --- | --- |
| **D3DXMath Function** | **DirectXMath Equivalent** |
| D3DXBoxBoundProbe | [**BoundingBox::Intersects(XMVECTOR, XMVECTOR, float&)**](http://msdn.microsoft.com/en-us/library/windows/desktop/hh437817(v=vs.85).aspx) |
| D3DXComputeBoundingBox | [**BoundingBox::CreateFromPoints**](http://msdn.microsoft.com/en-us/library/windows/desktop/hh437802(v=vs.85).aspx) |
| D3DXComputeBoundingSphere | [**BoundingSphere::CreateFromPoints**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.directxcollision.boundingsphere.createfrompoints(v=vs.85).aspx) |
| D3DXSphereBoundProbe | [**BoundingSphere::Intersects(XMVECTOR, XMVECTOR, float&)**](http://msdn.microsoft.com/en-us/library/windows/desktop/hh437818(v=vs.85).aspx) |
| D3DXIntersectTriFunction | [**TriangleTests::Intersects**](http://msdn.microsoft.com/en-us/library/windows/desktop/hh855857(v=vs.85).aspx) |
| D3DXFloat32To16Array | [**XMConvertFloatToHalfStream**](http://msdn.microsoft.com/en-us/library/windows/desktop/hh437913(v=vs.85).aspx) |
| D3DXFloat16To32Array | [**XMConvertHalfToFloatStream**](http://msdn.microsoft.com/en-us/library/windows/desktop/hh437922(v=vs.85).aspx) |
| D3DXVec2Length | [**XMVector2Length**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector2length(v=vs.85).aspx) or [**XMVector2LengthEst**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector2lengthest(v=vs.85).aspx) |
| D3DXVec2LengthSq | [**XMVector2LengthSq**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector2lengthsq(v=vs.85).aspx) |
| D3DXVec2Dot | [**XMVector2Dot**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector2dot(v=vs.85).aspx) |
| D3DXVec2CCW | [**XMVector2Cross**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector2cross(v=vs.85).aspx) |
| D3DXVec2Add | [**XMVectorAdd**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectoradd(v=vs.85).aspx) |
| D3DXVec2Subtract | [**XMVectorSubtract**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectorsubtract(v=vs.85).aspx) |
| D3DXVec2Minimize | [**XMVectorMin**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectormin(v=vs.85).aspx) |
| D3DXVec2Maximize | [**XMVectorMax**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectormax(v=vs.85).aspx) |
| D3DXVec2Scale | [**XMVectorScale**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectorscale(v=vs.85).aspx) |
| D3DXVec2Lerp | [**XMVectorLerp**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorlerp(v=vs.85).aspx) or [**XMVectorLerpV**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorlerpv(v=vs.85).aspx) |
| D3DXVec2Normalize | [**XMVector2Normalize**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector2normalize(v=vs.85).aspx) or [**XMVector2NormalizeEst**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector2normalizeest(v=vs.85).aspx) |
| D3DXVec2Hermite | [**XMVectorHermite**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorhermite(v=vs.85).aspx) or [**XMVectorHermiteV**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorhermitev(v=vs.85).aspx) |
| D3DXVec2CatmullRom | [**XMVectorCatmullRom**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorcatmullrom(v=vs.85).aspx) or [**XMVectorCatmullRomV**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorcatmullromv(v=vs.85).aspx) |
| D3DXVec2BaryCentric | [**XMVectorBaryCentric**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorbarycentric(v=vs.85).aspx) or [**XMVectorBaryCentricV**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorbarycentricv(v=vs.85).aspx) |
| D3DXVec2Transform | [**XMVector2Transform**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.transformation.xmvector2transform(v=vs.85).aspx) |
| D3DXVec2TransformCoord | [**XMVector2TransformCoord**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.transformation.xmvector2transformcoord(v=vs.85).aspx) |
| D3DXVec2TransformNormal | [**XMVector2TransformNormal**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.transformation.xmvector2transformnormal(v=vs.85).aspx) |
| D3DXVec2TransformArray | [**XMVector2TransformStream**](http://msdn.microsoft.com/en-us/library/windows/desktop/hh404775(v=vs.85).aspx) |
| D3DXVec2TransformCoordArray | [**XMVector2TransformCoordStream**](http://msdn.microsoft.com/en-us/library/windows/desktop/hh404773(v=vs.85).aspx) |
| D3DXVec2TransformNormalArray | [**XMVector2TransformNormalStream**](http://msdn.microsoft.com/en-us/library/windows/desktop/hh404774(v=vs.85).aspx) |
| D3DXVec3Length | [**XMVector3Length**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector3length(v=vs.85).aspx) or [**XMVector3LengthEst**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector3lengthest(v=vs.85).aspx) |
| D3DXVec3LengthSq | [**XMVector3LengthSq**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector3lengthsq(v=vs.85).aspx) |
| D3DXVec3Dot | [**XMVector3Dot**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector3dot(v=vs.85).aspx) |
| D3DXVec3Cross | [**XMVector3Cross**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector3cross(v=vs.85).aspx) |
| D3DXVec3Add | [**XMVectorAdd**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectoradd(v=vs.85).aspx) |
| D3DXVec3Subtract | [**XMVectorSubtract**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectorsubtract(v=vs.85).aspx) |
| D3DXVec3Minimize | [**XMVectorMin**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectormin(v=vs.85).aspx) |
| D3DXVec3Maximize | [**XMVectorMax**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectormax(v=vs.85).aspx) |
| D3DXVec3Scale | [**XMVectorScale**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectorscale(v=vs.85).aspx) |
| D3DXVec3Lerp | [**XMVectorLerp**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorlerp(v=vs.85).aspx) or [**XMVectorLerpV**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorlerpv(v=vs.85).aspx) |
| D3DXVec3Normalize | [**XMVector3Normalize**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector3normalize(v=vs.85).aspx) or [**XMVector3NormalizeEst**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector3normalizeest(v=vs.85).aspx) |
| D3DXVec3Hermite | [**XMVectorHermite**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorhermite(v=vs.85).aspx) or [**XMVectorHermiteV**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorhermitev(v=vs.85).aspx) |
| D3DXVec3CatmullRom | [**XMVectorCatmullRom**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorcatmullrom(v=vs.85).aspx) or [**XMVectorCatmullRomV**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorcatmullromv(v=vs.85).aspx) |
| D3DXVec3BaryCentric | [**XMVectorBaryCentric**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorbarycentric(v=vs.85).aspx) or [**XMVectorBaryCentricV**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorbarycentricv(v=vs.85).aspx) |
| D3DXVec3Transform | [**XMVector3Transform**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.transformation.xmvector3transform(v=vs.85).aspx) |
| D3DXVec3TransformCoord | [**XMVector3TransformCoord**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.transformation.xmvector3transformcoord(v=vs.85).aspx) |
| D3DXVec3TransformNormal | [**XMVector3TransformNormal**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.transformation.xmvector3transformnormal(v=vs.85).aspx) |
| D3DXVec3TransformArray | [**XMVector3TransformStream**](http://msdn.microsoft.com/en-us/library/windows/desktop/hh404780(v=vs.85).aspx) |
| D3DXVec3TransformCoordArray | [**XMVector3TransformCoordStream**](http://msdn.microsoft.com/en-us/library/windows/desktop/hh404778(v=vs.85).aspx) |
| D3DXVec3TransformNormalArray | [**XMVector3TransformNormalStream**](http://msdn.microsoft.com/en-us/library/windows/desktop/hh404779(v=vs.85).aspx) |
| D3DXVec3Project | [**XMVector3Project**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.transformation.xmvector3project(v=vs.85).aspx) |
| D3DXVec3Unproject | [**XMVector3Unproject**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.transformation.xmvector3unproject(v=vs.85).aspx) |
| D3DXVec3ProjectArray | [**XMVector3ProjectStream**](http://msdn.microsoft.com/en-us/library/windows/desktop/hh404776(v=vs.85).aspx) |
| D3DXVec3UnprojectArray | [**XMVector3UnprojectStream**](http://msdn.microsoft.com/en-us/library/windows/desktop/hh404782(v=vs.85).aspx) |
| D3DXVec4Length | [**XMVector4Length**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector4length(v=vs.85).aspx) or [**XMVector4LengthEst**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector4lengthest(v=vs.85).aspx) |
| D3DXVec4LengthSq | [**XMVector4LengthSq**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector4lengthsq(v=vs.85).aspx) |
| D3DXVec4Dot | [**XMVector4Dot**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector4dot(v=vs.85).aspx) |
| D3DXVec4Add | [**XMVectorAdd**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectoradd(v=vs.85).aspx) |
| D3DXVec4Subtract | [**XMVectorSubtract**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectorsubtract(v=vs.85).aspx) |
| D3DXVec4Minimize | [**XMVectorMin**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectormin(v=vs.85).aspx) |
| D3DXVec4Maximize | [**XMVectorMax**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectormax(v=vs.85).aspx) |
| D3DXVec4Scale | [**XMVectorScale**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectorscale(v=vs.85).aspx) |
| D3DXVec4Lerp | [**XMVectorLerp**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorlerp(v=vs.85).aspx) or [**XMVectorLerpV**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorlerpv(v=vs.85).aspx) |
| D3DXVec4Cross | [**XMVector4Cross**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector4cross(v=vs.85).aspx) |
| D3DXVec4Normalize | [**XMVector4Normalize**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector4normalize(v=vs.85).aspx) or [**XMVector4NormalizeEst**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvector4normalizeest(v=vs.85).aspx) |
| D3DXVec4Hermite | [**XMVectorHermite**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorhermite(v=vs.85).aspx) or [**XMVectorHermiteV**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorhermitev(v=vs.85).aspx) |
| D3DXVec4CatmullRom | [**XMVectorCatmullRom**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorcatmullrom(v=vs.85).aspx) or [**XMVectorCatmullRomV**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorcatmullromv(v=vs.85).aspx) |
| D3DXVec4BaryCentric | [**XMVectorBaryCentric**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorbarycentric(v=vs.85).aspx) or [**XMVectorBaryCentricV**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorbarycentricv(v=vs.85).aspx) |
| D3DXVec4Transform | [**XMVector4Transform**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.transformation.xmvector4transform(v=vs.85).aspx) |
| D3DXVec4TransformArray | [**XMVector4TransformStream**](http://msdn.microsoft.com/en-us/library/windows/desktop/hh404783(v=vs.85).aspx) |
| D3DXMatrixIdentity | [**XMMatrixIdentity**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixidentity(v=vs.85).aspx) |
| D3DXMatrixDeterminant | [**XMMatrixDeterminant**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixdeterminant(v=vs.85).aspx) |
| D3DXMatrixDecompose | [**XMMatrixDecompose**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixdecompose(v=vs.85).aspx) |
| D3DXMatrixTranspose | [**XMMatrixTranspose**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixtranspose(v=vs.85).aspx) |
| D3DXMatrixMultiply | [**XMMatrixMultiply**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixmultiply(v=vs.85).aspx) |
| D3DXMatrixMultiplyTranspose | [**XMMatrixMultiplyTranspose**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixmultiplytranspose(v=vs.85).aspx) |
| D3DXMatrixInverse | [**XMMatrixInverse**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixinverse(v=vs.85).aspx) |
| D3DXMatrixScaling | [**XMMatrixScaling**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixscaling(v=vs.85).aspx) |
| D3DXMatrixTranslation | [**XMMatrixTranslation**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixtranslation(v=vs.85).aspx) |
| D3DXMatrixRotationX | [**XMMatrixRotationX**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixrotationx(v=vs.85).aspx) |
| D3DXMatrixRotationY | [**XMMatrixRotationY**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixrotationy(v=vs.85).aspx) |
| D3DXMatrixRotationZ | [**XMMatrixRotationZ**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixrotationz(v=vs.85).aspx) |
| D3DXMatrixRotationAxis | [**XMMatrixRotationAxis**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixrotationaxis(v=vs.85).aspx) |
| D3DXMatrixRotationQuaternion | [**XMMatrixRotationQuaternion**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixrotationquaternion(v=vs.85).aspx) |
| D3DXMatrixRotationYawPitchRoll | [**XMMatrixRotationRollPitchYaw**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixrotationrollpitchyaw(v=vs.85).aspx) (Note the order of parameters is different: D3DXMatrixRotationYawPitchRoll takes yaw, pitch, roll, **XMMatrixRotationRollPitchYaw** takes pitch, yaw, roll) |
| D3DXMatrixTransformation | [**XMMatrixTransformation**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixtransformation(v=vs.85).aspx) |
| D3DXMatrixTransformation2D | [**XMMatrixTransformation2D**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixtransformation2d(v=vs.85).aspx) |
| D3DXMatrixAffineTransformation | [**XMMatrixAffineTransformation**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixaffinetransformation(v=vs.85).aspx) |
| D3DXMatrixAffineTransformation2D | [**XMMatrixAffineTransformation2D**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixaffinetransformation2d(v=vs.85).aspx) |
| D3DXMatrixLookAtRH | [**XMMatrixLookAtRH**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixlookatrh(v=vs.85).aspx) |
| D3DXMatrixLookAtLH | [**XMMatrixLookAtLH**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixlookatlh(v=vs.85).aspx) |
| D3DXMatrixPerspectiveRH | [**XMMatrixPerspectiveRH**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixperspectiverh(v=vs.85).aspx) |
| D3DXMatrixPerspectiveLH | [**XMMatrixPerspectiveLH**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixperspectivelh(v=vs.85).aspx) |
| D3DXMatrixPerspectiveFovRH | [**XMMatrixPerspectiveFovRH**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixperspectivefovrh(v=vs.85).aspx) |
| D3DXMatrixPerspectiveFovLH | [**XMMatrixPerspectiveFovLH**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixperspectivefovlh(v=vs.85).aspx) |
| D3DXMatrixPerspectiveOffCenterRH | [**XMMatrixPerspectiveOffCenterRH**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixperspectiveoffcenterrh(v=vs.85).aspx) |
| D3DXMatrixPerspectiveOffCenterLH | [**XMMatrixPerspectiveOffCenterLH**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixperspectiveoffcenterlh(v=vs.85).aspx) |
| D3DXMatrixOrthoRH | [**XMMatrixOrthographicRH**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixorthographicrh(v=vs.85).aspx) |
| D3DXMatrixOrthoLH | [**XMMatrixOrthographicLH**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixorthographiclh(v=vs.85).aspx) |
| D3DXMatrixOrthoOffCenterRH | [**XMMatrixOrthographicOffCenterRH**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixorthographicoffcenterrh(v=vs.85).aspx) |
| D3DXMatrixOrthoOffCenterLH | [**XMMatrixOrthographicOffCenterLH**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixorthographicoffcenterlh(v=vs.85).aspx) |
| D3DXMatrixShadow | [**XMMatrixShadow**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixshadow(v=vs.85).aspx) |
| D3DXMatrixReflect | [**XMMatrixReflect**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.matrix.xmmatrixreflect(v=vs.85).aspx) |
| D3DXQuaternionLength | [**XMQuaternionLength**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionlength(v=vs.85).aspx) |
| D3DXQuaternionLengthSq | [**XMQuaternionLengthSq**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionlengthsq(v=vs.85).aspx) |
| D3DXQuaternionDot | [**XMQuaternionDot**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaterniondot(v=vs.85).aspx) |
| D3DXQuaternionIdentity | [**XMQuaternionIdentity**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionidentity(v=vs.85).aspx) |
| D3DXQuaternionIsIdentity | [**XMQuaternionIsIdentity**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionisidentity(v=vs.85).aspx) |
| D3DXQuaternionConjugate | [**XMQuaternionConjugate**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionconjugate(v=vs.85).aspx) |
| D3DXQuaternionToAxisAngle | [**XMQuaternionToAxisAngle**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaterniontoaxisangle(v=vs.85).aspx) |
| D3DXQuaternionRotationMatrix | [**XMQuaternionRotationMatrix**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionrotationmatrix(v=vs.85).aspx) |
| D3DXQuaternionRotationAxis | [**XMQuaternionRotationAxis**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionrotationaxis(v=vs.85).aspx) |
| D3DXQuaternionRotationYawPitchRoll | [**XMQuaternionRotationRollPitchYaw**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionrotationrollpitchyaw(v=vs.85).aspx) (Note the order of parameters is different: D3DXQuaternionRotationYawPitchRoll takes yaw, pitch, roll, **XMQuaternionRotationRollPitchYaw**takes pitch, yaw, roll) |
| D3DXQuaternionMultiply | [**XMQuaternionMultiply**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionmultiply(v=vs.85).aspx) |
| D3DXQuaternionNormalize | [**XMQuaternionNormalize**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionnormalize(v=vs.85).aspx) or [**XMQuaternionNormalizeEst**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionnormalizeest(v=vs.85).aspx) |
| D3DXQuaternionInverse | [**XMQuaternionInverse**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternioninverse(v=vs.85).aspx) |
| D3DXQuaternionLn | [**XMQuaternionLn**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionln(v=vs.85).aspx) |
| D3DXQuaternionExp | [**XMQuaternionExp**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionexp(v=vs.85).aspx) |
| D3DXQuaternionSlerp | [**XMQuaternionSlerp**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionslerp(v=vs.85).aspx) or [**XMQuaternionSlerpV**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionslerpv(v=vs.85).aspx) |
| D3DXQuaternionSquad | [**XMQuaternionSquad**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionsquad(v=vs.85).aspx) or [**XMQuaternionSquadV**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionsquadv(v=vs.85).aspx) |
| D3DXQuaternionSquadSetup | [**XMQuaternionSquadSetup**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionsquadsetup(v=vs.85).aspx) |
| D3DXQuaternionBaryCentric | [**XMQuaternionBaryCentric**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionbarycentric(v=vs.85).aspx) or [**XMQuaternionBaryCentricV**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.quaternion.xmquaternionbarycentricv(v=vs.85).aspx) |
| D3DXPlaneDot | [**XMPlaneDot**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.plane.xmplanedot(v=vs.85).aspx) |
| D3DXPlaneDotCoord | [**XMPlaneDotCoord**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.plane.xmplanedotcoord(v=vs.85).aspx) |
| D3DXPlaneDotNormal | [**XMPlaneDotNormal**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.plane.xmplanedotnormal(v=vs.85).aspx) |
| D3DXPlaneScale | [**XMVectorScale**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectorscale(v=vs.85).aspx) |
| D3DXPlaneNormalize | [**XMPlaneNormalize**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.plane.xmplanenormalize(v=vs.85).aspx) or [**XMPlaneNormalizeEst**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.plane.xmplanenormalizeest(v=vs.85).aspx) |
| D3DXPlaneIntersectLine | [**XMPlaneIntersectLine**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.plane.xmplaneintersectline(v=vs.85).aspx) |
| D3DXPlaneFromPointNormal | [**XMPlaneFromPointNormal**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.plane.xmplanefrompointnormal(v=vs.85).aspx) |
| D3DXPlaneFromPoints | [**XMPlaneFromPoints**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.plane.xmplanefrompoints(v=vs.85).aspx) |
| D3DXPlaneTransform | [**XMPlaneTransform**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.plane.xmplanetransform(v=vs.85).aspx) |
| D3DXPlaneTransformArray | [**XMPlaneTransformStream**](http://msdn.microsoft.com/en-us/library/windows/desktop/hh404689(v=vs.85).aspx) |
| D3DXColorNegative | [**XMColorNegative**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.color.xmcolornegative(v=vs.85).aspx) |
| D3DXColorAdd | [**XMVectorAdd**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectoradd(v=vs.85).aspx) |
| D3DXColorSubtract | [**XMVectorSubtract**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectorsubtract(v=vs.85).aspx) |
| D3DXColorScale | [**XMVectorScale**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.arithmetic.xmvectorscale(v=vs.85).aspx) |
| D3DXColorModulate | [**XMColorModulate**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.color.xmcolormodulate(v=vs.85).aspx) |
| D3DXColorLerp | [**XMVectorLerp**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorlerp(v=vs.85).aspx) or [**XMVectorLerpV**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.geometric.xmvectorlerpv(v=vs.85).aspx) |
| D3DXColorAdjustSaturation | [**XMColorAdjustSaturation**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.color.xmcoloradjustsaturation(v=vs.85).aspx) |
| D3DXColorAdjustContrast | [**XMColorAdjustContrast**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.color.xmcoloradjustcontrast(v=vs.85).aspx) |
| D3DXFresnelTerm | [**XMFresnelTerm**](http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.utilities.xmfresnelterm(v=vs.85).aspx) |

**Note**  [Spherical Harmonics](http://go.microsoft.com/fwlink/p/?LinkId=262885) functions for DirectXMath are available separately. There is no DirectXMath equivalent to**ID3DXMatrixStack**.

-

## 1. Шейдеры

At a very high level, data enters the graphics pipeline as a stream of primitives and is processed by up to as many as three shader stages:

* A vertex shader performs per-vertex processing such as transformations, skinning, vertex displacement, and calculating per-vertex material attributes. Tessellation of higher-order primitives should be done before the vertex shader executes. As a minimum, a vertex shader must output vertex position in homogeneous clip space. Optionally, the vertex shader can output texture coordinates, vertex color, vertex lighting, fog factors, and so on.
* A geometry shader performs per-primitive processing such as material selection and silhouette-edge detection, and can generate new primitives for point sprite expansion, fin generation, shadow volume extrusion, and single pass rendering to multiple faces of a cube texture.
* A pixel shader performs per-pixel processing such as texture blending, lighting model computation, and per-pixel normal and/or environmental mapping. Pixel shaders work in concert with vertex shaders; the output of a vertex shader provides the inputs for a pixel shader. In Direct3D 9 some pixel operations (such as fog blending, stencil operations, and render-target blending) occur after the pixel shader is finished.

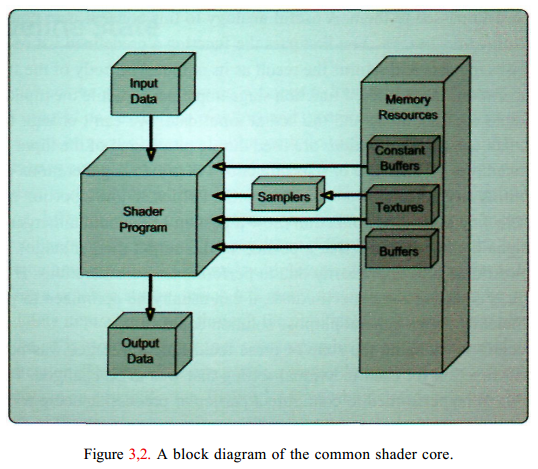
These stages are completely programmable using the High Level Shading Language ([HLSL](mk:@MSITStore:D:\Microsoft%20DirectX%20SDK%20(June%202010)\Documentation\DirectX9\windows_graphics.chm::/direct3dhlsl/dx_graphics_hlsl_reference.htm)). HLSL shaders can be compiled at author-time or at runtime, and set at runtime into the appropriate pipeline stage. Direct3D 9 shaders can be designed using [shader model 1](mk:@MSITStore:D:\Microsoft%20DirectX%20SDK%20(June%202010)\Documentation\DirectX9\windows_graphics.chm::/direct3dhlsl/dx_graphics_hlsl_sm1.htm), [shader model 2](mk:@MSITStore:D:\Microsoft%20DirectX%20SDK%20(June%202010)\Documentation\DirectX9\windows_graphics.chm::/direct3dhlsl/dx_graphics_hlsl_sm2.htm) and [shader model 3](mk:@MSITStore:D:\Microsoft%20DirectX%20SDK%20(June%202010)\Documentation\DirectX9\windows_graphics.chm::/direct3dhlsl/dx_graphics_hlsl_sm3.htm); Direct3D 10 shaders can only be designed on [shader model 4](mk:@MSITStore:D:\Microsoft%20DirectX%20SDK%20(June%202010)\Documentation\DirectX9\windows_graphics.chm::/direct3dhlsl/dx_graphics_hlsl_sm4.htm).

### Шейдерная архитектура в Direct3D 11

#### Общее шейдерное ядро (common shader core)

- все шейдеры имеют общую структуру, называемую общим шейдерным ядром (common shader core)

- общее шейдерное ядро предоставляет общую базовую функциональность для всех шейдеров: общее (generic) устройство (design) ввода и вывода стадии конвейера, общий интерфейс к ресурсам, которые шейдер может использовать



#### Архитектура шейдерного ядра

- перед выполнением шейдер проходит две стадии:

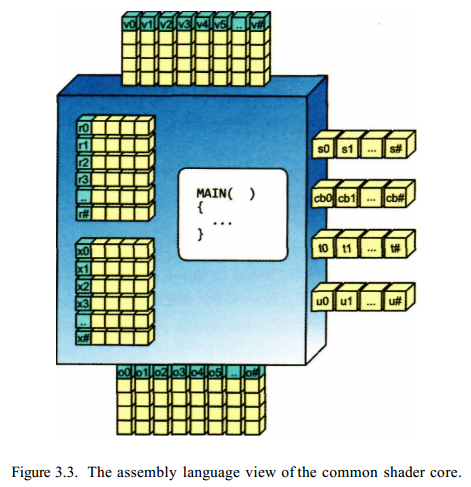
1 - шейдерная программа на HLSL компилируется в ассемблерный байт-код, **оперирующий векторными регистрами** (vector-register-based assembly language)

2- ассемблерный байт-код обрабатывается видео-драйвером в машинно-специфические инструкции, которые могут отличаться для разных графических процессоров

- байт-код даёт понимание работы шейдерного процессора

- для связи HLSL-программы с шейдерным байт-кодом, последний задаёт набор **регистров**, используемых компилятором

- эти **регистры** являются, в соновном, четырёхкомпонентными векторными регистрами, которые также могут использовать индивидуальные компоненты для предоставления функциональности скалярных регистров



**Регистры шейдерного ядра**

|  |  |  |  |
| --- | --- | --- | --- |
| **Регистры** | **Функция** | **Модифицируемость** | **Обозначение/Синтаксис** |
| входа | *получение входящих данных* | r | v# |
| временные | *хранение промежуточных вычислений* | rw | r#, x[n]# |
| текстурные | *для передачи тесктурных ресурсов через SRV* | r | t# |
| константно-буферные | *для связи с константными буферами ()* | b#, cb[n]# |
| семплерные | *для передачи семплеров (ID3D11SamplerState)* | s# |
| immidiate constant buffer |  | icb[index] |
| unordered access |  | rw | u# |
| выхода | *передача данных на следующую стадию конвейера* |  | o# |

- в ассемблерном представлении общее шейдерное ядро имеет следующий вид:

### Установка эффекта в Direct3D 11: шейдеры, константные буферы, текстуры

0. **Исходные данные**: ID3D11XXXShader\*, ID3D11Bob\*, ID3D11buffer\*, тип\_отражатель\_ cbuffer,

объект\_отражатель\_cbuffer;

1. **Компиляция шейдеров**: D3DcompileFromFile(ID3D11Blob\*\* - байт-код шейдера);

2. **Создание шейдеров**: device->CreateXXXShader(байт-код шейдера, … , ID3D11XXXShader\*\*);

3. **Создание формата вершин на основе байт-кода вершинного шейдера**:

device->CreateInputLayout(ID3D11Bob\*, …);

4. **Создание константных буферов**:

а) создание описателя D3D11\_BUFFER\_DESC desc;

б) заполнение описателя: в частности, поле ByteWidth = sizeof(тип\_отражатель\_ cbuffer)

opt) инициализация КБ:

- создать и заполнить объект\_отражатель\_cbuffer

- D3D11\_SUBRESOURCE\_DATA InitData;

- InitData.pSysMem = & объект\_отражатель\_cbuffer

в) создание КБ: device->CreateBuffer(&desc, [InitData]/NULL, ID3D11Buffer\*\*);

5. **Обновление константных буферов**:

devCon->Map(…); обновление через объект\_отражатель\_cbuffer; devCon->Umap();

6. **Установка объектов (шейдеров, КБ, текстур и т.д.)**:

- *Соотнесение КБ и текстур (ресурсов) с конкретным шейдером задаётся последовательностью установки шейдеров, т.е. связь КБ и текстур будет происходить с последним установленным шейдером. Следовательно шейдеры устанавливаются перед константными буферами и прочими ресурсами*

а) Шейдеры:

devCon->XXSetShader(ID3D11XXXShader\*, NULL, 0);

б) КБ:

devCon->XXSetConstantBuffers(**#\_индекс\_КБ\_регистра**, UINT NumBuffers, ID3D11Buffer\* const \* - &массив\_КБ);

*регистры КБ: b# и cb#*

в) Текстуры:

devCon- >XXSetShaderResources(**#\_индекс\_текстурного\_регистра**, кол\_во\_ресурсов, &SRV);

*текстурные регистры: t#*

где: XX – PS, VS, DS, HS, CS, GS

## 3.3 Composition of transformations

Suppose **S** is a scaling matrix, **R** is a rotation matrix, and **T** is a translation matrix. Assume we have a cube made up of eight vertices, **v***i*, for *i* = 0,1,…,7, and we wish to apply these three transformations to each vertex successively. The obvious way to do this is step-by-step:



However, because matrix multiplication is associative, we can instead write this equivalently as:



We can think of the matrix **C** = **SRT** as a matrix that encapsulates all three transformations into one net affine transformation matrix. In other words, matrix-matrix multiplication allows us to concatenate transforms.

This has performance implications. To see this, assume that a 3D object is composed of 20,000 points and that we want to apply these three successive geometric transformations to the object. Using the step-by-step approach, we would require 20,000×3 vector-matrix multiplications. On the other hand, using the combined matrix approach requires 20,000 vector-matrix multiplications and two matrix-matrix multiplications. Clearly, two extra matrix-matrix multiplications is a cheap price to pay for the large savings in vector-matrix multiplications.

|  |  |  |
| --- | --- | --- |
|  | **Note** | Again, we point out that matrix multiplication is not commutative. This is even seen geometrically. For example, a rotation followed by a translation, which we can describe by the matrix product **RT**, does not result in the same transformation as the same translation followed by the same rotation, that is, **TR**. [Figure 3.7](mk:@MSITStore:F:\DEVELOPMENT\GAME%20DEVELOPMENT\Engine\DirectX,%20C++\Books\Introduction%20to%203D%20Game%20Programming%20with%20DirectX%2010.chm::/31.html#ch03fig07) demonstrates this.    Figure 3.7: (a) Rotating first and then translating. (b) Translating first and then rotating. |



**Equation of matrix concatenation**

The preceding formula reflects the left-to-right rule of matrix concatenation. That is, the visible effects of the matrices that you use to create a composite matrix occur in left-to-right order.

## Функции для работы с векторами и матрицами

* FLOAT D3DXVec3Length( // Returns ||**v**||

CONST D3DXVECTOR3 \*pV); // Input **v**

* FLOAT D3DXVec3LengthSq( // Returns ||**v**||2

CONST D3DXVECTOR3 \*pV); // Input **v**

* FLOAT D3DXVec3Dot( // Returns **v**1 **· v**2

CONST D3DXVECTOR3 \*pV1*,* // Input **v**1

CONST D3DXVECTOR3 \*pV2); // Input **v**2

* D3DXVECTOR3 \*D3DXVec3Cross(

D3DXVECTOR3 \*pOut, // Returns **v**1 **× v**2

CONST D3DXVECTOR3 \*pV1, // Input **v**1

CONST D3DXVECTOR3 \*pV2); // Input **v**2

* D3DXVECTOR3 \*WINAPI D3DXVec3Normalize(

D3DXVECTOR3 \*pOut, // Returns **v** / ||**v**||

CONST D3DXVECTOR3 \*pV, // Input **v**

Функции для работы с матрицами:

* D3DXMATRIX \*D3DXMatrixIdentity(

D3DXMATRIX \*pOut); // Makes input the identity matrix.

* D3DXMATRIX \*D3DXMatrixTranspose(

D3DXMATRIX \*pOut, // Output **M***T*

CONST D3DXMATRIX \*pM); // Input **M**

* D3DXMATRIX \*D3DXMatrixInverse(

D3DXMATRIX \*pOut, // Output **M−**1

FLOAT \*pDeterminant, // Not needed, specify zero

CONST D3DXMATRIX \*pM); // Input **M**

* D3DXVECTOR4 \*D3DXVec4Transform(

D3DXVECTOR4 \*pOut, // Output **vM**

CONST D3DXVECTOR4 \*pV, // Input **v**

CONST D3DXMATRIX \*pM); // Input **M**

- для использования этих функций нужен модуль d3dx10.h :

#include <d3d10x.h>

// Constructs a scaling matrix:

D3DXMATRIX \*WINAPI D3DXMatrixScaling(

D3DXMATRIX \*pOut, // Returns **S**

FLOAT sx, FLOAT sy, FLOAT sz); // Scaling factors

// Constructs an x-axis rotation matrix:

D3DXMATRIX \*WINAPI D3DXMatrixRotationX(

D3DXMATRIX \*pOut, // Returns **R**n, where **n** = (1, 0, 0)

FLOAT Angle); // Angle *θ* to rotate

// Constructs a y-axis rotation matrix:

D3DXMATRIX \*WINAPI D3DXMatrixRotationY(

D3DXMATRIX \*pOut, // Returns **R**n, where **n** = (0, 1, 0)

FLOAT Angle); // Angle *θ* to rotate

// Constructs a z-axis rotation matrix:

D3DXMATRIX \*WINAPI D3DXMatrixRotationZ(

D3DXMATRIX \*pOut, // Returns **R**n, where **n** = (0, 0, 1)

FLOAT Angle); // Angle *θ* to rotate

// Constructs an arbitrary axis rotation matrix:

D3DXMATRIX \*WINAPI D3DXMatrixRotationAxis(

D3DXMATRIX \*pOut, // Returns **R**n

CONST D3DXVECTOR3 \*pV, // Axis **n** to rotate about

FLOAT Angle); // Angle *θ* to rotate

// Constructs a translation matrix:

D3DXMATRIX \*WINAPI D3DXMatrixTranslation(

D3DXMATRIX \*pOut, // Returns **T**

FLOAT x, FLOAT y, FLOAT z); // Translation factors

Direct3D – это низкоуровневый API (интерфейс прикладного программирования/программный интерфейс приложения), позволяющий создавать трёхмерную графику при помощи специального аппаратного обеспечения, такого как графические видеокарты со встроенным графическим акселератором.

В сущности, Direct3D предоставляет программные интерфейсы, при помощи которых можно управлять графическим аппартным обеспечением. Т.о. имея в виде интерфейса Direct3D слой абстракции между приложением и графическим АО, мы можем не безпокиться об особенностях последнего до тех пор пока оно является Direct3D совместимым оборудованием.

COM (Component Object Model – модель компонентных объектов) - стандартный механизм, включающий интерфейсы, с помощью которых одни объекты предоставляют свои сервисы другим, - является основой многих объектных технологий, в том числе OLE и ActiveX. В конетксте применения COM для разработки приложений DirectX технологию COM можно рассматривать как средство, позволяющее DirectX быть независимым от языка программирования и иметь обратную совместимость. Обычно о COM объекте говорят как об интерфейсе, который программистом приложений DirectX может рассматриваться как класс C++. Большая часть деталей реализации COM скрыта от программиста приложений DirectX. Единственная информация о COM, которая важна для разработчика DirectX приложения, это то, что работа ведётся не самим COM интефейсом, как объектом класса, создаваемом при помощи ключевого слова new, а с указателем на этот интерфейс, который можно получить, используя специальные функции или методы другого COM интерфейса. Идентификаторы имён COM интерфейсов всегда начинаются с префикса I, например ID3D10ShaderResourceView.

## 4.1.3 Textures and Data Resource Formats

A 2D texture is a matrix of data elements. One use for 2D textures is to store 2D image data, where each element in the texture stores the color of a pixel. However, this is not the only usage; for example, in an advanced technique called normal mapping, each element in the texture stores a 3D vector instead of a color. Therefore, although it is common to think of textures as storing image data, they are really more general purpose than that. A 1D texture is like a 1D array of data elements, and a 3D texture is like a 3D array of data elements. As will be discussed in later chapters, textures are more than just arrays of data; they can have mipmap levels, and the GPU can do special operations on them, such as applying filters and multisampling. In addition, a texture cannot store arbitrary kinds of data; it can only store certain kinds of data formats, which are described by the DXGI\_FORMAT enumerated type. Some example formats are:

* DXGI\_FORMAT\_R32G32B32\_FLOAT: Each element has three 32-bit floating-point components.
* DXGI\_FORMAT\_R16G16B16A16\_UNORM: Each element has four 16-bit components mapped to the [0, 1] range.
* DXGI\_FORMAT\_R32G32\_UINT: Each element has two 32-bit unsigned integer components.
* DXGI\_FORMAT\_R8G8B8A8\_UNORM: Each element has four 8-bit unsigned components mapped to the [0, 1] range.
* DXGI\_FORMAT\_R8G8B8A8\_SNORM: Each element has four 8-bit signed components mapped to the [−1, 1] range.
* DXGI\_FORMAT\_R8G8B8A8\_SINT: Each element has four 8-bit signed integer components mapped to the [−128, 127] range.
* DXGI\_FORMAT\_R8G8B8A8\_UINT: Each element has four 8-bit unsigned integer components mapped to the [0, 255] range.

Note that the R, G, B, A letters are used to stand for red, green, blue, and alpha, respectively. Colors are formed as combinations of the basis colors red, green, and blue (e.g., equal values of red and green make yellow). The alpha channel or alpha component is generally used to control transparency. However, as we said earlier, textures need not store color information; for example, the format

DXGI\_FORMAT\_R32G32B32\_FLOAT

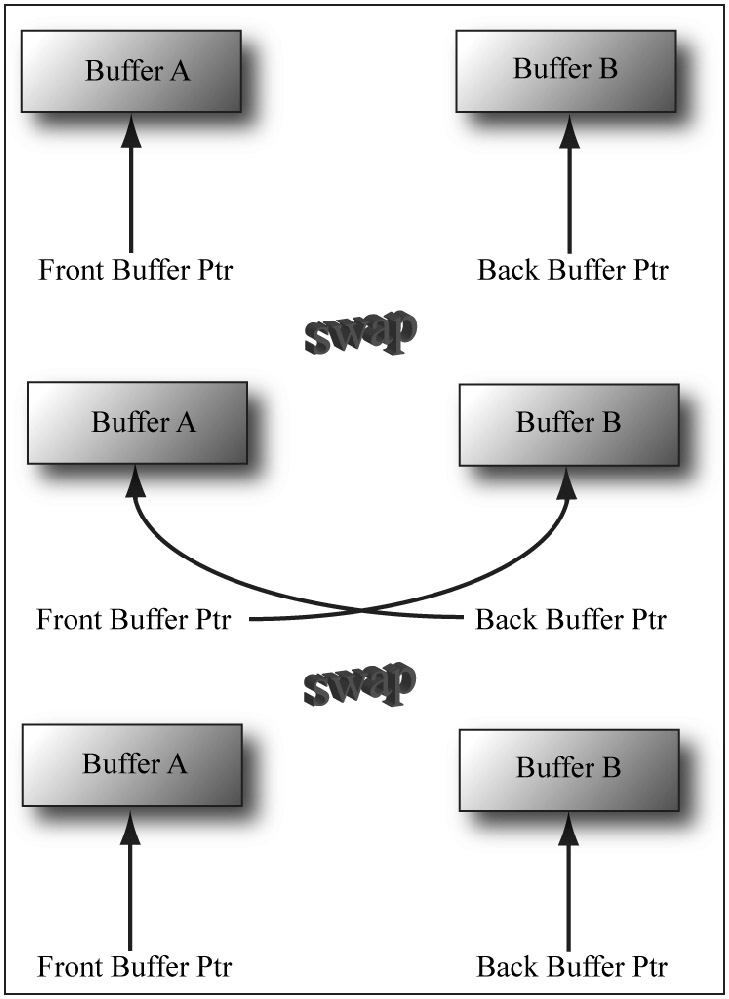
has three floating-point components and can therefore store a 3D vector with floating-point coordinates. There are also *typeless* formats, where we just reserve memory and then specify how to reinterpret the data at a later time (sort of like a cast) when the texture is bound to the pipeline; for example, the following typeless format reserves elements with four 8-bit components, but does not specify the data type (e.g., integer, floating-point, unsigned integer):

DXGI\_FORMAT\_R8G8B8A8\_TYPELESS

В широком смысле текстура в DirectX это контейнер для хранения данных заданного типа, т.е. массив. Такой массив можно использовать как для хранения цветовой информации, так, например, и для задания вектора или набора векторов. Нетипизированные (typless) форматы текстур позволяют производить хранение данных разных типов и преобразовывать их во время подключения к графическому конвейеру.

## 4.1.4 The Swap Chain and Page Flipping

Для избежания мерцания выводимого на экран изображения используются два текстурных буффера (т.е. две текстуры) – передний (front buffer) и задний (back buffer) буферы. Кадр изображения записывается в задний буффер (где он хранится как текстура) и во время вертикального интервала гашения (интервала обратного хода луча по вертикали) целиком выводится на экран, т.о. наблюдатель видит целые кадры, а не то, как кадр отрисовывается. В переднем буфере хранится графическая информация, которая отображается на экране в данный момент времени, тогда, как следующий кадр анимации записывается в задний буфер. После того, как кадр был записан в задний буфер, оба буфера – передний и задний меняются ролями. Такая смена ролей буферов называется презентацией (presenting). Презентация является эффективной операцией, поскольку указатели на текущие передний и задний буферы просто меняются адресами.



Сперва производится рендеринг в буффер В, служащий текущим задним буфером. Как только кадр готов, указатели меняются адресами и буфер В становится передним буфером, а буфер А становится новым задним буфером. Дальше следующий кадр рендерится в буфер А. Как тоько кадр готов, указатели меняются адресами – буфер А становится передним буфером, а буфер В снова становится задним буфером.

- передний и задний буферы (и механизм смены их ролей) образуют swap chain

- в Direct3D swap chain представлена интерфейсом IDXGISwapChain

- этот интерфейс хранит текстуры переднего и заднего буферов, а также предоставляет методы для изменения их размеров и презентации (presenting)

- использование двух буферов (переднего и заднего) называется двойной буферизацией

- могут быть использованы более чем два буфера - тройная буферизация – применение трёх буферов;

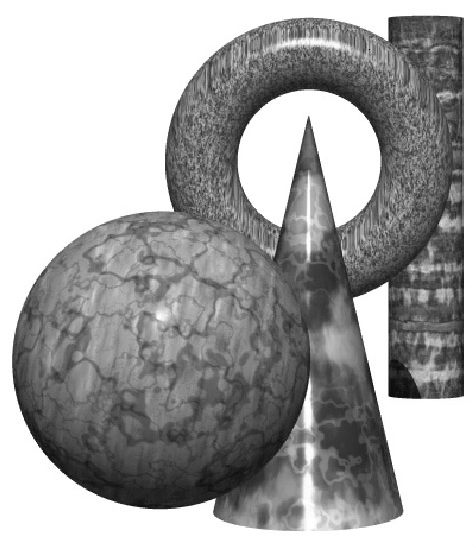
- однако двух буферов, как правило, бывает достаточно

- хотя задний буфер является текстурой (а значит элемент должен называться текселем), часто элемент называется пикселем, т.к. в случае заднего буфера, в нём (в элементе) хранится информация о цвете; иногда элементы текстуры могут называть пикселями, даже если в них не хранится цвет (e.g. pixels of normal map)

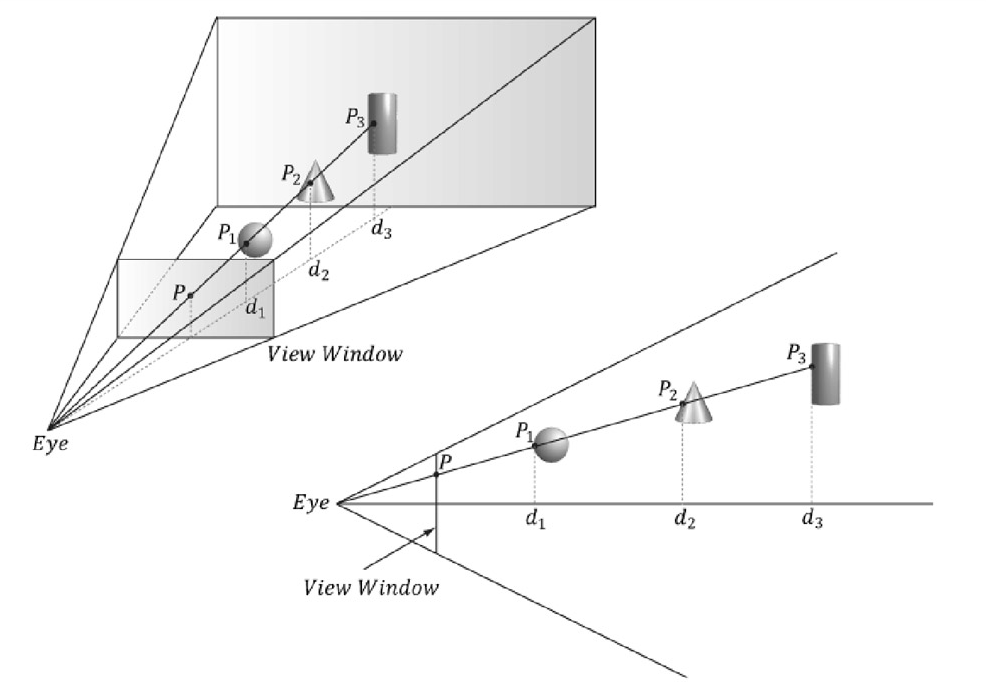
## 4.1.5 Depth Buffering

The *depth buffer* is an example of a texture that does not contain image data, but rather depth information about a particular pixel. The possible depth values range from 0.0 to 1.0, where 0.0 denotes the closest an object can be to the viewer and 1.0 denotes the farthest an object can be from the viewer. There is a one-to-one correspondence between each element in the depth buffer and each pixel in the back buffer (i.e., the *ij*th element in the back buffer corresponds to the *ij*th element in the depth buffer). So if the back buffer had a resolution of 1280×1024, there would be 1280×1024 depth entries.

Let us emphasize that with depth buffering, the order in which we draw the objects does not matter.



|  |  |
| --- | --- |
| Remark | To handle the depth problem, one might suggest drawing the objects in the scene in the order of farthest to nearest. In this way, near objects will be painted over far objects, and the correct results should be rendered. This is how a painter would draw a scene. However, this method has its own problems — sorting a large data set and intersecting geometry. Besides, the graphics hardware gives us depth buffering for free. |



The view window corresponds to the 2D image (back buffer) we generate of the 3D scene. We see that three different pixels can be projected to the pixel P. Intuition tells us that P1 should be written to P since it is closer to the viewer and blocks the other two pixels. The depth buffer algorithm provides a mechanical procedure for determining this on a computer. **Note that we show the depth values relative to the 3D scene being viewed, but they are actually normalized to the range [0.0, 1.0] when stored in the depth buffer.**

| **Operation** | ***P*** | ***d*** | **Description** |
| --- | --- | --- | --- |
| Clear operation | Black | 1.0 | Pixel and corresponding depth entry initialized. |
| Draw cylinder | *P*3 | *d*3 | Since *d*3 ≤ *d* = 1.0, the depth test passes and we update the buffers by setting *P* = *P*3 and *d* = *d*3. |
| Draw sphere | *P*1 | *d*1 | Since *d*1 ≤ *d* = *d*3, the depth test passes and we update the buffers by setting *P* = *P*1 and *d* = *d*1. |
| Draw cone | *P*1 | *d*1 | Since d2 > *d* = *d*1, the depth test fails and we do not update the buffers. |

As you can see, we only update the pixel and its corresponding depth value in the depth buffer when we find a pixel with a smaller depth value. In this way, after all is said and done, the pixel that is closest to the viewer will be the one rendered. (You can try switching the drawing order around and working through this example again if you are still not convinced.)

To summarize, **depth buffering works by computing a depth value for each pixel and performing a depth test.** The *depth test* compares the depths of pixels competing to be written to a particular pixel location on the back buffer. The pixel with the depth value closest to the viewer wins, and that is the pixel that gets written to the back buffer. This makes sense because the pixel closest to the viewer obscures the pixels behind it.

**The depth buffer is a texture, so it must be created with certain data formats.** The formats used for depth buffering are as follows:

* DXGI\_FORMAT\_D32\_FLOAT\_S8X24\_UINT: Specifies a 32-bit floating-point depth buffer, with 8 bits (unsigned integer) reserved for the stencil buffer mapped to the [0, 255] range and 24 bits used for padding.
* DXGI\_FORMAT\_D32\_FLOAT: Specifies a 32-bit floating-point depth buffer.
* DXGI\_FORMAT\_D24\_UNORM\_S8\_UINT: Specifies an unsigned 24-bit depth buffer mapped to the [0, 1] range with 8 bits (unsigned integer) reserved for the stencil buffer mapped to the [0, 255] range.
* DXGI\_FORMAT\_D16\_UNORM: Specifies an unsigned 16-bit depth buffer mapped to the [0, 1] range.

|  |  |  |
| --- | --- | --- |
|  | **Note** | An application is not required to have a stencil buffer, but if it does, the **stencil buffer is always attached to the depth buffer**. For example, the 32-bit format  DXGI\_FORMAT\_D24\_UNORM\_S8\_UINT  uses 24 bits for the depth buffer and 8 bits for the stencil buffer. For this reason, **the depth buffer** **is better called the depth/stencil buffer**. Using the stencil buffer is a more advanced topic and will be explained in [Chapter 9](mk:@MSITStore:F:\DEVELOPMENT\GAME%20DEVELOPMENT\Engine\DirectX,%20C++\Books\Introduction%20to%203D%20Game%20Programming%20with%20DirectX%2010.chm::/105.html). |

## 4.1.6 Texture Resource Views

- ресурсы (такие как текстуры) присоединяются к конвейеру не напрямую, для этого DX использует ресурсные представления (resource view)

- посредством ресурсного представления текстуру можно присоединить к разным стадиям графического конвейера

- можно использовать т-ру как цель отрисовки (render target), т.е. для осуществления в неё отрисовки; также можно передать т-ру в шейдер (присоединить её к шейдерной стадии конвейера для текстурирования поверхностей объектов); текстурный ресурс, создаваемый сразу для двух этих случаев, будет иметь следующие флаги привязки (bind flags): D3D10\_BIND\_RENDER\_TARGET | D3D10\_BIND\_SHADER\_RESOURCE , что указывает на те две стадии конвейера, к которым будет привязана текстура

- использование ресурсных представлений связано с соображениями эффективности

- ресурсные представления используются D3D для определения к какой стадии конвейера будет привязана т-ра

- если, например, т-ра будет использована как цель отрисовки или как шейдерный ресурс, то потребуется создать два ресурсных представления: ID3D10RenderTargetView и ID3D10ShaderResourceView

- для ресурсов чей формат был указан как нетипизированный (typeless), необходимо указать тип во время создания ресурсного представления

- thus, with typeless formats, it is possible for the elements of a texture to be viewed as floating-point values in one pipeline stage and as integers in another

- creating a fully-typed resource restricts the resource to the format it was created with. This enables the runtime to optimize access. Therefore, you should only create a typeless resource if you really need it; otherwise, create a fully typed resource.

## 4.1.7 Multisampling

Direct3D supports an antialiasing technique called *multisampling*, which works by taking the neighboring pixels into consideration when computing the final color of a pixel. Thus, the technique is called multisampling because it uses multiple pixel samples to compute the final color of a pixel.

Структура DXGI\_SAMPLE\_DESC описывает настройки мультисемплинга, которые потребуется установить во время создания swap chain буфера и бэкбуфера:

typedef struct DXGI\_SAMPLE\_DESC {

UINT Count;

UINT Quality;

} DXGI\_SAMPLE\_DESC, \*LPDXGI\_SAMPLE\_DESC;

The Count member specifies the number of samples to take per pixel, and the Quality member is used to specify the desired quality level. A higher quality is more expensive, so a trade-off between quality and speed must be made. The range of quality levels depends on the texture format and the number of samples to take per pixel. Use the following method to query the number of quality levels for a given texture format and sample count:

HRESULT ID3D10Device::CheckMultisampleQualityLevels(

DXGI\_FORMAT Format, UINT SampleCount, UINT \*pNumQualityLevels);

This method returns 0 (zero) if the format and sample count combination is not supported by the device. Other wise, the number of quality levels for the given combination will be returned through the pNumQualityLevels parameter. Valid quality levels for a texture format and sample count combination range from 0 to pNumQualityLevels −1.

Как задний буфер так и буфер глубины должны создаваться с одниковыми настройками мультисемплинга

Чтобы отключить мультисемплинг нужно установить следующие значения полей описателя DXGI\_SAMPLE\_DESC: Count в 1 и Quality в 0.

## 4.2 Инициализация Direct3D

- Если компилировать в режиме Release, то:

- добавить следующее d3d11.lib;d3dx11.lib; в Linker->Input->Additional Dependecies ()

- отключить Precompiled Headers

- Для включения всех библиотек в исполняемый файл (статическая линковка) нужно: C/C++ ->Code Generation->Runtime Library: Multi-threaded (/MT) или Multi-threaded Debug (/MTd) для Release и Debug конфигураций соответственно

- для отладки release-версии нужно:

- убрать оптимизацию: C/C++ -> Optimization->Optimization: Disabled (изначально: Maximize Speed /O2)

- выставить ключ для генерации отладочной информации: Linker->Debugging->Generate Debug Info: Yes(/DEBUG)

- также можно не отключать оптимизацию, но тогда отладчик будет перемещаться по коду безсвязно, будут отсутствовать многие переменные и т.д.

d3d10.lib – находится в %DXSDK\_DIR%\Lib\x86 и в %DXSDK\_DIR%\Lib\x64

#pragma comment(lib,"d3d10.lib") и #pragma comment (lib, "d3dx10.lib") – дают указание компоновщику подключить d3d10.lib и d3dx10.lib

#include <d3d10.h>

Пути к .lib файлам должны быть прописаны в свойствах или файлы должны лежать в каталоге проекта, тогда ссылку на их местоположение нужно убрать из путей (VC++ Directories).

Также вместо pragma можно указать d3d10.lib в ветке Linker->Input->Additional Dependencies, ввести d3d10.lib и отделить его от других ссылок точкой с запятой. Это будет эквивалентным способом указать компоновщику подключить d3d10.lib. При этом в Linker->General->Additional Library Directories также должен быть указан путь к d3d10.lib .

Если d3d10.lib лежит в каталоге проекта, то пути указывать не нужно (ни в VC++ Directories ни в Linker), достаточно будет подключить d3d10.lib через окно свойств проекта или при помощи pragma.

Если d3d10.lib будет указан в путях подключения и одновременно будет находиться в каталоге проекта, то компоновщик не сможет выбрать какой из двух файлов подключать и возникнет ошибка «error LNK2019: unresolved external symbol \_D3D10CreateDeviceAndSwapChain@32 referenced in function F - *функция, где вызывается D3D10CreateDeviceAndSwapChain()*»

(На 64-битной ОС подключается библиотека d3d10.lib из %DXSDK\_DIR%\Lib\x64)

Для переноса проекта нужно все внешние файлы, на которые он ссылается скопировать в католог проекта.

1)

Describe the characteristics of the swap chain we are going to create by filling out an instance of the DXGI\_SWAP\_CHAIN\_DESC structure. Create the ID3D10Device and IDXGISwapChain interfaces using the D3D10CreateDeviceAndSwapChain function.

Создаём Direct3D устройство и цепь перестановок (swap chain) - ID3D10Device и IDXGISwapChain интерфейсы

2)

Create a render target view to the swap chain’s back buffer.

Создаём render target view для заднего буфера цепи перестановок (для бекбуфера)

3)

Create the depth/stencil buffer and its associated depth/stencil view. Bind the render target view and depth/stencil view to the output merger stage of the rendering pipeline so that they can be used by Direct3D.

Создание буфера глубины и трафарета и его представления. Привязка представления к OM стадии графического конвейера

4)

Set the viewport.

Установка области вывода (viewport)

The following subsections show how to initialize Direct3D. Our process of initializing Direct3D can be broken down into the following steps:

1. Describe the characteristics of the swap chain we are going to create by filling out an instance of the DXGI\_SWAP\_CHAIN\_DESC structure.

DXGI\_SWAP\_CHAIN\_DESC sd;

sd.BufferDesc.Width = mClientWidth; // use window's client area dims

sd.BufferDesc.Height = mClientHeight;

sd.BufferDesc.RefreshRate.Numerator = 60;

sd.BufferDesc.RefreshRate.Denominator = 1;

sd.BufferDesc.Format = DXGI\_FORMAT\_R8G8B8A8\_UNORM;

sd.BufferDesc.ScanlineOrdering = DXGI\_MODE\_SCANLINE\_ORDER\_UNSPECIFIED;

sd.BufferDesc.Scaling = DXGI\_MODE\_SCALING\_UNSPECIFIED;

// No multisampling.

sd.SampleDesc.Count = 1;

sd.SampleDesc.Quality = 0;

sd.BufferUsage = DXGI\_USAGE\_RENDER\_TARGET\_OUTPUT;

sd.BufferCount = 1;

sd.OutputWindow = mhMainWnd;

sd.Windowed = true;

sd.SwapEffect = DXGI\_SWAP\_EFFECT\_DISCARD;

sd.Flags = 0;

1. Create the ID3D10Device and IDXGISwapChain interfaces using the D3D10CreateDeviceAndSwapChain function.

The ID3D10Device interface is the chief Direct3D interface and can be thought of as our software controller of the physical graphics device hardware; that is, through this interface we can interact with the hardware and instruct it to do things (such as clear the back buffer, bind resources to the various pipeline stages, and draw geometry).

HRESULT WINAPI D3D10CreateDeviceAndSwapChain(

IDXGIAdapter \*pAdapter,

D3D10\_DRIVER\_TYPE DriverType,

HMODULE Software,

UINT Flags,

UINT SDKVersion,

DXGI\_SWAP\_CHAIN\_DESC \*pSwapChainDesc,

IDXGISwapChain \*\*ppSwapChain,

ID3D10Device \*\*ppDevice);

* pAdapter: Specifies the display adapter we want the created device to represent. Specifying null for this parameter uses the primary display adapter. We always use the primary adapter in the sample programs of this book.
* DriverType: In general, you will always specify D3D10\_DRIVER\_TYPE\_HARDWARE for this parameter to use 3D hardware acceleration for rendering. Specifying D3D10\_DRIVER\_TYPE\_REFERENCE creates a so-called reference device. The *reference device* is a software implementation of Direct3D with the goal of correctness (it is extremely slow since it is a software implementation). There are two reasons to use the reference device:
  + To test code your hardware does not support; for example, to test Direct3D 10.1 code when you do not have a Direct3D 10.1-capable graphics card.
  + To test for driver bugs. If you have code that works correctly with the reference device, but not with the hardware, then there is probably a bug in the hardware drivers.
* Software: This is used for supplying a software rasterizer. We always specify null since we are using hardware for rendering. Moreover, one must have a software rasterizer available in order to use one.
* Flags: Optional device creation flags. For release mode builds, this will generally be 0 (no extra flags); for debug mode builds, this should be D3D10\_CREATE\_DEVICE\_DEBUG to enable the debug layer. When the debug flag is specified, Direct3D will send debug messages to the VC++ output window;
* SDKVersion: Always specify D3D10\_SDK\_VERSION.
* pSwapChainDesc: A pointer to the filled out DXGI\_SWAP\_CHAIN\_DESC structure describing the swap chain we want to create.
* ppSwapChain: Returns the created swap chain.
* ppDevice: Returns the created device.

Here is an example call of this function:

DXGI\_SWAP\_CHAIN\_DESC sd;

/\* Initialize sd \*/

UINT createDeviceFlags = 0;

#if defined(DEBUG) || defined(\_DEBUG)

createDeviceFlags |= D3D10\_CREATE\_DEVICE\_DEBUG;

#endif

ID3D10Device\* md3dDevice;

IDXGISwapChain\* mSwapChain;

D3D10CreateDeviceAndSwapChain(0, D3D10\_DRIVER\_TYPE\_HARDWARE,

0, createDeviceFlags, D3D10\_SDK\_VERSION,

&sd, &mSwapChain, &md3dDevice);

1. Create a render target view to the swap chain’s back buffer.

We do not bind a resource to a pipeline stage directly; instead, we must create a resource view to the resource and bind the view to the pipeline stage. In particular, in order to bind the back buffer to the output merger stage of the pipeline (so Direct3D can render onto it), we need to create a render target view to the back buffer. The following example code shows how this is done:

ID3D10RenderTargetView\* mRenderTargetView;

ID3D10Texture2D\* backBuffer;

mSwapChain->GetBuffer(0, \_\_uuidof(ID3D10Texture2D),

reinterpret\_cast<void\*\*>(&backBuffer));

md3dDevice->CreateRenderTargetView(backBuffer, 0, &mRenderTargetView);

ReleaseCOM(backBuffer);

* A pointer to the swap chain’s back buffer is obtained using the IDXGISwapChain::GetBuffer method. The first parameter of this method is an index identifying the particular back buffer we want to get (in case there is more than one). In our demos, we only use one back buffer, and it has index 0. The second parameter is the inter face type of the buffer, which is usually always a 2D texture (ID3D10Texture2D). The third parameter returns a pointer to the back buffer.
* To create the render target view, we use the ID3D10Device::CreateRenderTargetView method. The first parameter specifies the resource that will be used as the render target, which, in the example above, is the back buffer. The second parameter is a pointer to a D3D10\_RENDER\_TARGET\_VIEW\_DESC. Among other things, this structure describes the data type of the elements in the resource. If the resource was created with a typed format (i.e., not typeless), then this parameter can be null, which indicates to use the format the resource was created with. The third parameter returns a pointer to the create render target view object.
* The call to IDXGISwapChain::GetBuffer increases the COM reference count to the back buffer, which is why we release it (ReleaseCOM) at the end of the code fragment.

1. Create the depth/stencil buffer and its associated depth/stencil view.

As described in [§4.1.5](mk:@MSITStore:F:\DEVELOPMENT\GAME%20DEVELOPMENT\Engine\DirectX,%20C++\Books\Introduction%20to%203D%20Game%20Programming%20with%20DirectX%2010.chm::/37.html#188), the depth buffer is just a 2D texture that stores the depth information (and stencil information if using stenciling). To create a texture, we need to fill out a D3D10\_TEXTURE2D\_DESC structure describing the texture to create, and then call the ID3D10Device::CreateTexture2D method.

typedef struct D3D10\_TEXTURE2D\_DESC {

UINT Width;

UINT Height;

UINT MipLevels;

UINT ArraySize;

DXGI\_FORMAT Format;

DXGI\_SAMPLE\_DESC SampleDesc;

D3D10\_USAGE Usage;

UINT BindFlags;

UINT CPUAccessFlags;

UINT MiscFlags;

} D3D10\_TEXTURE2D\_DESC;

* Width: The width of the texture in texels.
* Height: The height of the texture in texels.
* MipLevels: The number of mipmap levels. Mipmaps are covered in [Chapter 7](mk:@MSITStore:F:\DEVELOPMENT\GAME%20DEVELOPMENT\Engine\DirectX,%20C++\Books\Introduction%20to%203D%20Game%20Programming%20with%20DirectX%2010.chm::/79.html), “[Texturing](mk:@MSITStore:F:\DEVELOPMENT\GAME%20DEVELOPMENT\Engine\DirectX,%20C++\Books\Introduction%20to%203D%20Game%20Programming%20with%20DirectX%2010.chm::/79.html).” For creating the depth/stencil buffer, our texture only needs one mipmap level.
* ArraySize: The number of textures in a texture array. For the depth/stencil buffer, we only need one texture.
* Format: A member of the DXGI\_FORMAT enumerated type specifying the format of the texels. For a depth/stencil buffer, this needs to be one of the formats shown in [§4.1.5](mk:@MSITStore:F:\DEVELOPMENT\GAME%20DEVELOPMENT\Engine\DirectX,%20C++\Books\Introduction%20to%203D%20Game%20Programming%20with%20DirectX%2010.chm::/37.html#188):
* DXGI\_FORMAT\_D32\_FLOAT\_S8X24\_UINT: Specifies a 32-bit floating-point depth buffer, with 8 bits (unsigned integer) reserved for the stencil buffer mapped to the [0, 255] range and 24 bits used for padding.
* DXGI\_FORMAT\_D32\_FLOAT: Specifies a 32-bit floating-point depth buffer.
* DXGI\_FORMAT\_D24\_UNORM\_S8\_UINT: Specifies an unsigned 24-bit depth buffer mapped to the [0, 1] range with 8 bits (unsigned integer) reserved for the stencil buffer mapped to the [0, 255] range.
* DXGI\_FORMAT\_D16\_UNORM: Specifies an unsigned 16-bit depth buffer mapped to the [0, 1] range.
* SampleDesc: set the sample count to 1 and the quality level to 0 (to indicate there will not be multisampling).
* Usage: A member of the D3D10\_USAGE enumerated type specifying how the texture will be used. The four usage values are:
  + D3D10\_USAGE\_DEFAULT: Specify this usage if the GPU (graphics processing unit) will be reading and writing to the resource. The CPU cannot read or write to a resource with this usage. For the depth/stencil buffer, we specify D3D10\_USAGE\_DEFAULT since the GPU will be doing all the reading and writing to the depth/stencil buffer.
  + D3D10\_USAGE\_IMMUTABLE: Specify this usage if the content of a resource never changes after creation. This allows for some potential optimizations, as the resource will be read-only by the GPU. The CPU cannot write to an immutable resource, except at creation time to initialize the resource. The CPU cannot read from an immutable resource.
  + D3D10\_USAGE\_DYNAMIC: Specify this usage if the application (CPU) needs to update the data contents of the resource frequently (e.g., on a per-frame basis). A resource with this usage can be read by the GPU and written to by the CPU.
  + D3D10\_USAGE\_STAGING: Specify this usage if the application (CPU) needs to be able to read a copy of the resource (i.e., the resource supports copying data from video memory to system memory).
* BindFlags: One or more flags ORed together, specifying where the resource will be bound to the pipeline. For a depth/stencil buffer, this needs to be D3D10\_BIND\_DEPTH\_STENCIL. Some other bind flags for textures are:
  + D3D10\_BIND\_RENDER\_TARGET: The texture will be bound as a render target to the pipeline.
  + D3D10\_BIND\_SHADER\_RESOURCE: The texture will be bound as a shader resource to the pipeline.
* CPUAccessFlags: Specifies how the CPU will access the resource. If the CPU needs to write to the resource, specify D3D10\_CPU\_ACCESS\_WRITE. A resource with write access must have usage D3D10\_USAGE\_DYNAMIC or D3D10\_USAGE\_STAGING. If the CPU needs to read from the buffer, specify D3D10\_CPU\_ACCESS\_READ. A buffer with read access must have usage D3D10\_USAGE\_STAGING. For the depth/stencil buffer, only the GPU writes and reads to the depth/buffer; therefore, we can specify 0 for this value, as the CPU will not be reading or writing to the depth/stencil buffer.
* MiscFlags: Optional flags, which do not apply to the depth/stencil buffer, so set to 0.

D3D10\_TEXTURE2D\_DESC depthStencilDesc;

depthStencilDesc.Width = mClientWidth;

depthStencilDesc.Height = mClientHeight;

depthStencilDesc.MipLevels = 1;

depthStencilDesc.ArraySize = 1;

depthStencilDesc.Format = DXGI\_FORMAT\_D24\_UNORM\_S8\_UINT;

depthStencilDesc.SampleDesc.Count = 1; // multisampling must match

depthStencilDesc.SampleDesc.Quality = 0; // swap chain values.

depthStencilDesc.Usage = D3D10\_USAGE\_DEFAULT;

depthStencilDesc.BindFlags = D3D10\_BIND\_DEPTH\_STENCIL;

depthStencilDesc.CPUAccessFlags = 0;

depthStencilDesc.MiscFlags = 0;

ID3D10Texture2D\* mDepthStencilBuffer;

ID3D10DepthStencilView\* mDepthStencilView;

HR(md3dDevice->CreateTexture2D(

&depthStencilDesc, 0, &mDepthStencilBuffer));

HR(md3dDevice->CreateDepthStencilView(

mDepthStencilBuffer, 0, &mDepthStencilView));

The second parameter of CreateTexture2D is a pointer to initial data to fill the texture with. However, since this texture is to be used as the depth/stencil buffer, we do not need to fill it ourselves with any data. Direct3D will write to the depth/stencil buffer directly when performing depth buffering and stencil operations. Thus, we specify null for the second parameter.

1. Bind the render target view and depth/stencil view to the output merger stage of the rendering pipeline so that they can be used by Direct3D.

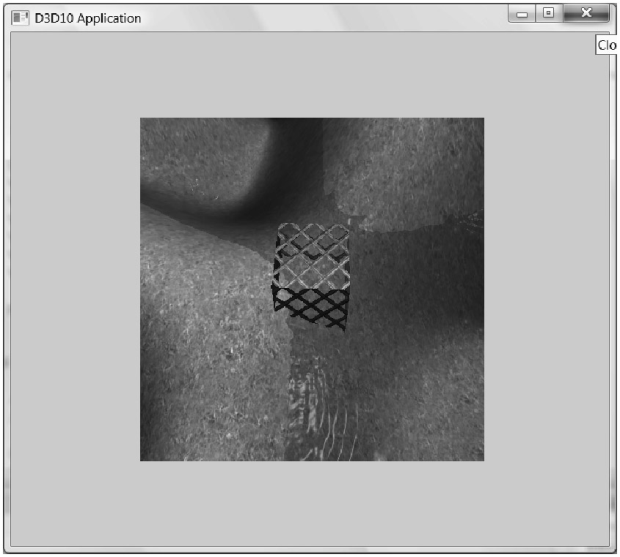
md3dDevice->OMSetRenderTargets(1, &RTV, DSV);

The first parameter is the number of render targets we are binding; we bind only one here, but more can be bound to render simultaneously to several render targets (an advanced technique). The second parameter is a pointer to the first element in an array of render target view pointers to bind to the pipeline. The third parameter is a pointer to the depth/stencil view to bind to the pipeline.

We can set an array of render target views, but only one depth/stencil view.

1. Set the viewport.

Usually we like to draw the 3D scene to the entire back buffer. However, sometimes we only want to draw the 3D scene into a subrectangle of the back buffer, as shown in figure:



The subrectangle of the back buffer we draw into is called the *viewport* and it is described by the following structure:

typedef struct D3D10\_VIEWPORT {

INT TopLeftX;

INT TopLeftY;

UINT Width;

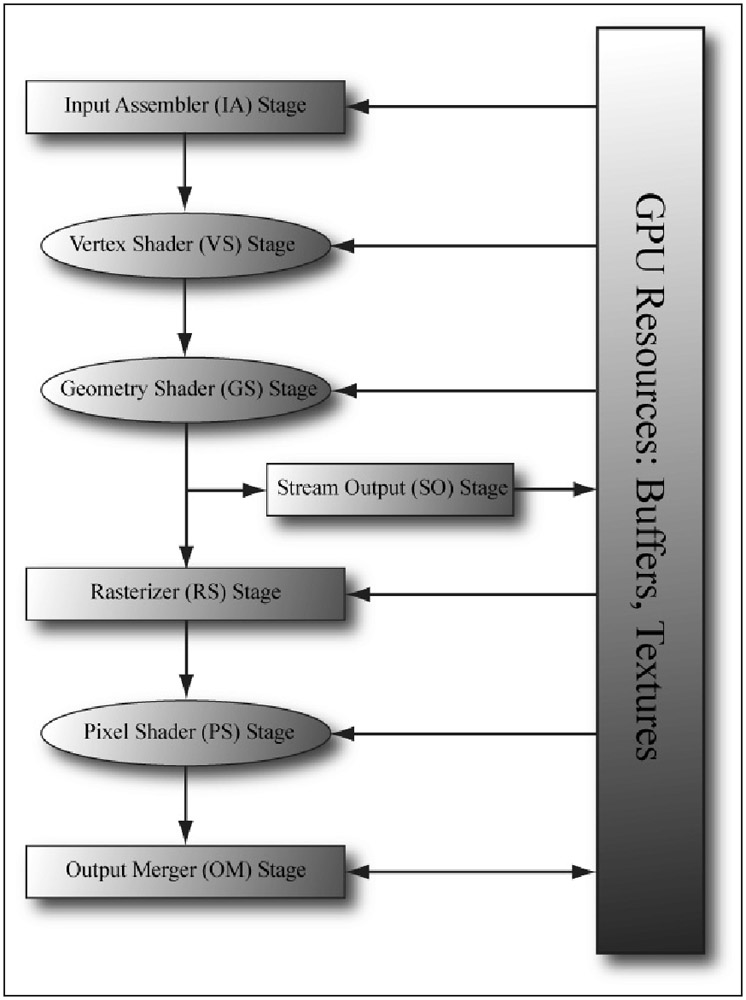
UINT Height;

FLOAT MinDepth;

FLOAT MaxDepth;

} D3D10\_VIEWPORT;

## 5. Обзор стадий графического конвейера



### 5.1 Компоновщик исходных данных Input Assembler (IA)

На стадии первичной компоновки данных конвейер Direct3D занимается считыванием из памяти геометрических данных – вершин и вершинных индексов - на основании установленных перед этим параметров, инструкций, определяющих детали проведения этого процесса. Сперва нужно задать какими будут вершины объектов, будут ли они просто представлять собой пространственные точки, содержащие координаты или же дополнительно будут использованы другие данные: векторы нормали, текстурные координаты. Для дальнейшего создания объектов – на следующих стадиях конвейера - потребуются списки вершин заданного формата и списки индексов, которые понадобятся для указания в каком порядке соединять эти вершины, чтобы получить желаемый геометрический объект. Для хранения и описания последних двух списков в D3D используются вершинные и индексные буферы.

The purpose of the input-assembler stage is to read primitive data (points, lines and/or triangles) from user-filled buffers and assemble the data into primitives that will be used by the other pipeline stages. The IA stage can assemble vertices into several different [primitive types](http://msdn.microsoft.com/en-us/library/windows/desktop/bb205124(v=vs.85).aspx)(such as line lists, triangle strips, or primitives with adjacency). New primitive types (such as a line list with adjacency or a triangle list with adjacency) have been added to support the geometry shader.

Adjacency information is visible to an application only in a geometry shader. If a geometry shader were invoked with a triangle including adjacency, for instance, the input data would contain 3 vertices for each triangle and 3 vertices for adjacency data per triangle.

### 5.2 Input layouts (описатели применяемых форматов вершин)

Direct3D позволяет создавать собственный формат вершин и хранить его описание в массиве структур D3D10\_INPUT\_ELEMENT\_DESC, который затем используется для создания описательного интерфейса ID3D10InputLayout. Затем этот интерфейс привязывается к пайплайну D3D и применяется для рендеринга всех объектов. Его действие будет актуальным до тех пор, пока его не заменят на новый.

Например, мы хотим применять два типа вершин – первый будет описывать геометрическое положение вершины в пространстве и связанный с ней цвет, второй – положение, нормаль в данной точке, текстурные координаты:

struct Vertex1

{

D3DXVECTOR3 pos;

D3DXCOLOR color;

};

struct Vertex2

{

D3DXVECTOR3 pos;

D3DXVECTOR3 normal;

D3DXVECTOR2 texC;

};

Каждый тип вершин нужно зарегистрировать в системе, для чего сперва заполняются массивы описательных структур D3D10\_INPUT\_ELEMENT\_DESC, где каждый элемент соответствует элементу вершинного типа:

typedef struct D3D10\_INPUT\_ELEMENT\_DESC {

LPCSTR SemanticName;

UINT SemanticIndex;

DXGI\_FORMAT Format;

UINT InputSlot;

UINT AlignedByteOffset

D3D10\_INPUT\_CLASSIFICATION InputSlotClass;

UINT InstanceDataStepRate;

} D3D10\_INPUT\_ELEMENT\_DESC;

D3D10\_INPUT\_ELEMENT\_DESC desc1[] =

{

{"POSITION", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 0,

D3D10\_INPUT\_PER\_VERTEX\_DATA, 0},

{"COLOR", 0, DXGI\_FORMAT\_R32G32B32A32\_FLOAT, 0, 12,

D3D10\_INPUT\_PER\_VERTEX\_DATA, 0}

};

D3D10\_INPUT\_ELEMENT\_DESC desc2[] =

{

{"POSITION", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 0,

D3D10\_INPUT\_PER\_VERTEX\_DATA, 0},

{"NORMAL", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 12,

D3D10\_INPUT\_PER\_VERTEX\_DATA, 0},

{"TEXCOORD", 0, DXGI\_FORMAT\_R32G32\_FLOAT, 0, 24,

D3D10\_INPUT\_PER\_VERTEX\_DATA, 0}

};

Затем на основе заполненных описателей создаётся объект ID3D10InputLayout :

HRESULT ID3D10Device::CreateInputLayout(

const D3D10\_INPUT\_ELEMENT\_DESC \*pInputElementDescs,

UINT NumElements,

const void \*pShaderBytecodeWithInputSignature,

SIZE\_T BytecodeLength,

ID3D10InputLayout \*\*ppInputLayout);

ID3D10Effect\* mFX;

ID3D10EffectTechnique\* mTech;

ID3D10InputLayout\* mVertexLayout;

/\* ...create the effect... \*/

mTech = mFX->GetTechniqueByName("ColorTech");

D3D10\_PASS\_DESC PassDesc;

mTech->GetPassByIndex(0)->GetDesc(&PassDesc);

HR(md3dDevice->CreateInputLayout(vertexDesc, 3,

PassDesc.pIAInputSignature, PassDesc.IAInputSignatureSize,

&mVertexLayout));

Далее необходимо связать полученный объект с конвейером Direct3D :

ID3D10InputLayout\* mVertexLayout;

/\* ...create the input layout... \*/

md3dDevice->IASetInputLayout(mVertexLayout);

Если в процессе рендеринга нужно сменить описатель применяемого формата вершин, то создаётся новый объект ID3D10InputLayout и регистрируется в системе, заменяя предыдущий:

md3dDevice->IASetInputLayout(mVertexLayout1);

/\* ...draw objects using input layout 1... \*/

md3dDevice->IASetInputLayout(mVertexLayout2);

/\* ...draw objects using input layout 2... \*/

### 5.3 Вершинный буфер (Vertex buffer)

Каждый меш хранится в виде массива вершин заданного формата (ID3D10InputLayout), но для того, чтобы GPU мог получить доступ к этим вершинам необходимо поместить их в вершинный буфер, ресурс D3D, который аналогично другим создаётся на основании описателей и затем связывается с пайплайном Direct3D.

Для создания ВБ нужно заполнить две следующие структуры-описателя:

- D3D10\_BUFFER\_DESC (опсиывает параметры самого буфера)

typedef struct D3D10\_BUFFER\_DESC {

UINT ByteWidth;

D3D10\_USAGE Usage;

UINT BindFlags;

UINT CPUAccessFlags;

UINT MiscFlags;

} D3D10\_BUFFER\_DESC;

- D3D10\_SUBRESOURCE\_DATA (описывает данные, которыми будет инициализирован ВБ)

typedef struct D3D10\_SUBRESOURCE\_DATA {

const void \*pSysMem;

UINT SysMemPitch;

UINT SysMemSlicePitch;

} D3D10\_SUBRESOURCE\_DATA;

pSysMem – указатель на массив, содержащий вершины, которыми будет заполнен ВБ;

два остальных параметра в ВБ не используются.

Пример создания вершинног буфера и связывания с конвейером D3D:

Vertex vertices[] =

{

{D3DXVECTOR3(-1.0f, -1.0f, -1.0f), WHITE},

{D3DXVECTOR3(-1.0f, +1.0f, -1.0f), BLACK},

{D3DXVECTOR3(+1.0f, +1.0f, -1.0f), RED},

{D3DXVECTOR3(+1.0f, -1.0f, -1.0f), GREEN},

{D3DXVECTOR3(-1.0f, -1.0f, +1.0f), BLUE},

{D3DXVECTOR3(-1.0f, +1.0f, +1.0f), YELLOW},

{D3DXVECTOR3(+1.0f, +1.0f, +1.0f), CYAN},

{D3DXVECTOR3(+1.0f, -1.0f, +1.0f), MAGENTA},

};

D3D10\_BUFFER\_DESC vbd;

vbd.Usage = D3D10\_USAGE\_IMMUTABLE;

vbd.ByteWidth = sizeof(Vertex) \* 8;

vbd.BindFlags = D3D10\_BIND\_VERTEX\_BUFFER;

vbd.CPUAccessFlags = 0;

vbd.MiscFlags = 0;

D3D10\_SUBRESOURCE\_DATA vinitData;

vinitData.pSysMem = vertices;

ID3D10Buffer\* mVB;

HR(md3dDevice->CreateBuffer(

&vbd, // description of buffer to create

&vinitData, // data to initialize buffer with

&mVB)); // return the created buffer

Связываем ВБ с пайплайном - передаём данные вершин в один из слотов конвейера (его индекс указывается в первом параметре; всего имеется 16 слотов с индексами 0-15):

UINT stride = sizeof(Vertex1);

UINT offset = 0;

md3dDevice->IASetVertexBuffers(0, 1, &mVB, &stride, &offset);

* StartSlot: The input slot to start binding vertex buffers to. There are 16 input slots indexed as 0-15.
* NumBuffers: The number of vertex buffers we are binding to the input slots, starting at StartSlot.
* ppVertexBuffers: Pointer to the first element of an array of vertex buffers.
* pStrides: Pointer to the first element of an array of strides (one for each vertex buffer and the *i*th stride corresponds to the *i*th vertex buffer). A *stride* is the size, in bytes, of an element in the corresponding vertex buffer.
* pOffsets: Pointer to the first element of an array of offsets (one for each vertex buffer and the *i*th offset corresponds to the *i*th vertex buffer). This is an offset, in bytes, from the start of the vertex buffer to the position in the vertex buffer from which the input assembly should start reading the data. You would use this if you wanted to skip over some data at the front of the vertex buffer.

UINT stride = sizeof(Vertex1);

массив страйдов для каждого ВБ, где значение элемента равно размеру в байтах элемента соответствующего вершинного буфера;

если ВБ один, то массив страйдов вырождается до скаляра;

в метод передаётся указатель на первый элемент массива страйдов

UINT offset = 0; // массив смещений для каждого ВБ, где значение элемента равно

// размеру смещения в байтах от начала соответствующего вершинного буфера;

// если ВБ один, то массив смещений вырождается до скаляра;

// в метод передаётся указатель на первый элемент массива офсетов;

// смещение - количество байт от начала ВБ до элемента, с которого сборщик (IA) начнёт читать данные;

// имеет смысл тогда, когда одним ВБ задаётся сразу несколько объектов, тогда для каждого нужно задать смещение

Для отрисовки вершин используется следующий метод:

void ID3D10Device::Draw(UINT VertexCount, UINT StartVertexLocation);

Теперь пример на использование двух разных ВБ для разных типов вершин:

ID3D10Buffer\* mVB1; // stores vertices of type Vertex1

ID3D10Buffer\* mVB2; // stores vertices of type Vertex2

/\*...Create the vertex buffers...\*/

UINT stride = sizeof(Vertex1);

UINT offset = 0;

md3dDevice->IASetVertexBuffers(0, 1, &mVB1, &stride, &offset);

/\* ...draw objects using vertex buffer 1... \*/

stride = sizeof(Vertex2);

offset = 0;

md3dDevice->IASetVertexBuffers(0, 1, &mVB2, &stride, &offset);

/\* ...draw objects using vertex buffer 2... \*/

### 5.4 Индексный буфер (Index buffer)

ИБ используется для хранения списка индексов вершин объекта и создаётся аналогично ВБ:

DWORD indexList[24] = {

0, 1, 2, // Triangle 0

0, 2, 3, // Triangle 1

0, 3, 4, // Triangle 2

0, 4, 5, // Triangle 3

0, 5, 6, // Triangle 4

0, 6, 7, // Triangle 5

0, 7, 8, // Triangle 6

0, 8, 1 // Triangle 7

};

// Describe the index buffer we are going to create. Observe the

// D3D10\_BIND\_INDEX\_BUFFER bind flag

D3D10\_BUFFER\_DESC ibd;

ibd.Usage = D3D10\_USAGE\_IMMUTABLE;

ibd.ByteWidth = sizeof(DWORD) \* 24;

ibd.BindFlags = D3D10\_BIND\_INDEX\_BUFFER;

ibd.CPUAccessFlags = 0;

ibd.MiscFlags = 0;

// Specify the data to initialize the index buffer.

D3D10\_SUBRESOURCE\_DATA iinitData;

iinitData.pSysMem = indexList;

// Create the index buffer.

ID3D10Buffer\* mIB;

HR(md3dDevice->CreateBuffer(&ibd, &iinitData, &mIB));

Связывание с конвейером D3D :

md3dDevice->IASetIndexBuffer(mIB, DXGI\_FORMAT\_R32\_UINT, 0);

Для активации индексов в системе вместо метода Draw используется метод DrawIndexed() :

void ID3D10Device::DrawIndexed(

UINT IndexCount,

UINT StartIndexLocation,

INT BaseVertexLocation);

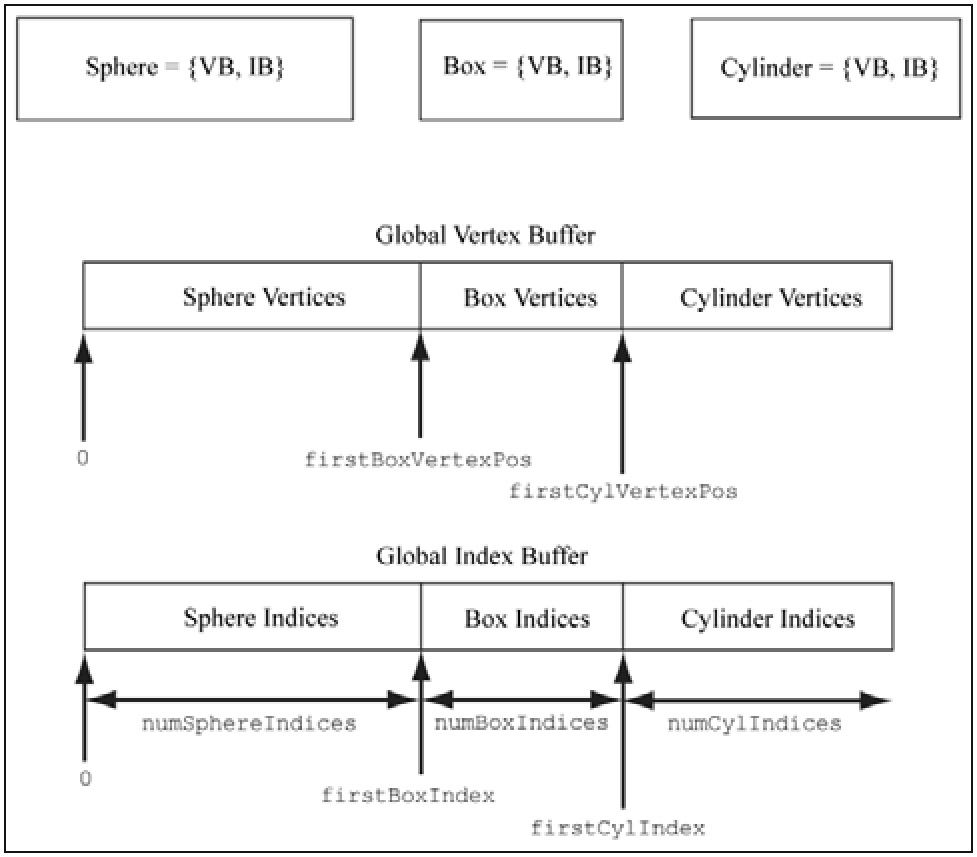
Если слить несколько ВБ и ИБ для разных объектов в один глобальный ВБ и ИБ соответственно, то позиции и индексы вершин сместятся. Для учёта этого применяются параметры StartIndexLocation и BaseVertexLocation :

например, мы слили в одно целое сферу, куб и цилиндр, тогда:

md3dDevice->DrawIndexed(numSphereIndices, 0, 0);

md3dDevice->DrawIndexed(numBoxIndices, firstBoxIndex, firstBoxVertexPos);

md3dDevice->DrawIndexed(numCylIndices, firstCylIndex, firstCylVertexPos);



### 5.6.5 Constant Buffers

Above the example vertex shader in the previous section was the code:

cbuffer cbPerObject

{

float4x4 gWVP;

};

This code defines a cbuffer object (constant buffer) called cbPerObject. *Constant buffers* are just blocks of data that can store different variables that may be accessed by a shader. In this example, the constant buffer stores a single 4×4 matrix called gWVP, representing the combined world, view, and projection matrices used to transform a point from local space to homogeneous clip space. In HLSL, a 4×4 matrix is declared by the built-in float4x4 type; to declare a 3×4 matrix and a 2×2 matrix, for example, you would use the float3x4 and float2x2 types, respectively. Data in constant buffers does not vary per vertex, but through the effects framework ([§5.13](mk:@MSITStore:F:\DEVELOPMENT\GAME%20DEVELOPMENT\Engine\Books\Frank%20D.Luna%20-%20Introduction%20to%203D%20Game%20Programming%20with%20DirectX%2010.chm::/58.html#395)), the C++ application code can update the contents of a constant buffer at run time. This provides a means for the C++ application code and the effect code to communicate. For instance, because the world matrix varies per object, the combined world, view, and projection matrix varies per object; therefore, when using the above vertex shader to draw multiple objects, we would need to update the gWVP variable appropriately before drawing each object.

The general advice is to create constant buffers based on the frequency in which you need to update their contents. For instance, you may create the following constant buffers:

cbuffer cbPerObject

{

float4x4 gWVP;

};

cbuffer cbPerFrame

{

float3 gLightDirection;

float3 gLightPosition;

float4 gLightColor;

};

cbuffer cbRarely

{

float4 gFogColor;

float gFogStart;

float gFogEnd;

};

In this example, we use three constant buffers. The first constant buffer stores the combined world, view, and projection matrix. This variable depends on the object, so it must be updated on a per-object basis. That is, if we are rendering 100 objects per frame, then we will be updating this variable 100 times per frame. The second constant buffer stores scene light variables. Here, we are assuming that the lights are animated and so need to be updated once every frame of animation. The last constant buffer stores variables used to control fog. Here, we are assuming that the scene fog rarely changes (e.g., maybe it only changes at certain times of day in the game).

The motivation for dividing up the constant buffers is efficiency. When a constant buffer is updated, all its variables must be updated; therefore, it is efficient to group them based on their update frequency to minimize redundant updates.

The Direct3D 10 and higher pipeline contains three programmable-shader stages (the rounded blocks in the [pipeline functional diagram](http://msdn.microsoft.com/en-us/library/windows/desktop/bb205123(v=vs.85).aspx)):

* [Vertex-Shader Stage](http://msdn.microsoft.com/en-us/library/windows/desktop/bb205146(v=vs.85).aspx#Vertex_Shader_Stage)
* [Geometry-Shader Stage](http://msdn.microsoft.com/en-us/library/windows/desktop/bb205146(v=vs.85).aspx#Geometry_Shader_Stage)
* [Pixel-Shader Stage](http://msdn.microsoft.com/en-us/library/windows/desktop/bb205146(v=vs.85).aspx#Pixel_Shader_Stage)

Each shader stage exposes its own unique functionality, built on the shader model 4.0 [common-shader core](http://msdn.microsoft.com/en-us/library/windows/desktop/bb509580(v=vs.85).aspx).

##### **Vertex-Shader Stage**

The vertex-shader stage processes vertices, typically performing operations such as transformations, skinning, and lighting. A vertex shader always takes a single input vertex and produces a single output vertex.

The vertex-shader (VS) stage processes vertices from the input assembler, performing per-vertex operations such as transformations, skinning, morphing, and per-vertex lighting. Vertex shaders always operate on a single input vertex and produce a single output vertex. The vertex shader stage must always be active for the pipeline to execute. If no vertex modification or transformation is required, a pass-through vertex shader must be created and set to the pipeline.

Each vertex shader input vertex can be comprised of up to 16 32-bit vectors (up to 4 components each) and each output vertex can be comprised of as many as 16 32-bit 4-component vectors. All vertex shaders must have a minimum of one input and one output, which can be as little as one scalar value.

The vertex-shader stage can consume two system generated values from the input assembler: VertexID and InstanceID (see System Values and Semantics). Since VertexID and InstanceID are both meaningful at a vertex level, and IDs generated by hardware can only be fed into the first stage that understands them, these ID values can only be fed into the vertex-shader stage.

Vertex shaders are always run on all vertices, including adjacent vertices in input primitive topologies with adjacency. The number of times that the vertex shader has been executed can be queried from the CPU using the VSInvocations pipeline statistic.

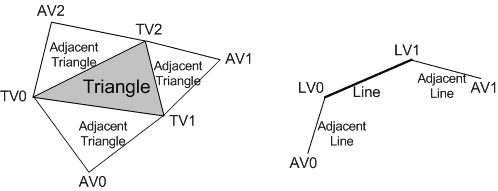
A vertex shader can perform load and texture sampling operations where screen-space derivatives are not required (using HLSL intrinsic functions: [Sample (DirectX HLSL Texture Object)](http://msdn.microsoft.com/en-us/library/windows/desktop/bb509695(v=vs.85).aspx), [SampleCmpLevelZero (DirectX HLSL Texture Object)](http://msdn.microsoft.com/en-us/library/windows/desktop/bb509697(v=vs.85).aspx), and [SampleGrad (DirectX HLSL Texture Object)](http://msdn.microsoft.com/en-us/library/windows/desktop/bb509698(v=vs.85).aspx)).

##### **Geometry-Shader Stage**

The geometry-shader stage processes entire primitives. Its input is a full primitive (which is three vertices for a triangle, two vertices for a line, or a single vertex for a point). In addition, each primitive can also include the vertex data for any edge-adjacent primitives. This could include at most an additional three vertices for a triangle or an additional two vertices for a line. The Geometry Shader also supports limited geometry amplification and de-amplification. Given an input primitive, the Geometry Shader can discard the primitive, or emit one or more new primitives.

Vertex positions leaving the geometry shader must be transformed to homogeneous clip space.

The geometry-shader (GS) stage runs application-specified shader code with vertices as input and the ability to generate vertices on output. Unlike vertex shaders, which operate on a single vertex, the geometry shader's inputs are the vertices for a full primitive (two vertices for lines, three vertices for triangles, or single vertex for point). Geometry shaders can also bring in the vertex data for the edge-adjacent primitives as input (an additional two vertices for a line, an additional three for a triangle). The following illustration shows a triangle and a line with adjacent vertices.



|  |  |
| --- | --- |
| TV | Triangle vertex |
| AV | Adjacent vertex |
| LV | Line vertex |

The geometry-shader stage can consume the SV\_PrimitiveID [system-generated value](http://msdn.microsoft.com/en-us/library/windows/desktop/bb205118(v=vs.85).aspx) that is auto-generated by the IA. This allows per-primitive data to be fetched or computed if desired.

The geometry-shader stage is capable of outputting multiple vertices forming a single selected topology (GS stage output topologies available are: tristrip, linestrip, and pointlist). The number of primitives emitted can vary freely within any invocation of the geometry shader, though the maximum number of vertices that could be emitted must be declared statically. Strip lengths emitted from a geometry shader invocation can be arbitrary, and new strips can be created via the [RestartStrip](http://msdn.microsoft.com/en-us/library/windows/desktop/bb509660(v=vs.85).aspx) HLSL function.

Geometry shader output may be fed to the rasterizer stage and/or to a vertex buffer in memory via the stream output stage. Output fed to memory is expanded to individual point/line/triangle lists (exactly as they would be passed to the rasterizer).

When a geometry shader is active, it is invoked once for every primitive passed down or generated earlier in the pipeline. Each invocation of the geometry shader sees as input the data for the invoking primitive, whether that is a single point, a single line, or a single triangle. A triangle strip from earlier in the pipeline would result in an invocation of the geometry shader for each individual triangle in the strip (as if the strip were expanded out into a triangle list). All the input data for each vertex in the individual primitive is available (i.e. 3 vertices for triangle), plus adjacent vertex data if applicable/available.

A geometry shader outputs data one vertex at a time by appending vertices to an output stream object. The topology of the streams is determined by a fixed declaration, choosing one of: PointStream, LineStream, or TriangleStream as the output for the GS stage. There are three types of stream objects available, PointStream, LineStream and TriangleStream which are all templated objects. The topology of the output is determined by their respective object type, while the format of the vertices appended to the stream is determined by the template type. Execution of a geometry shader instance is atomic from other invocations, except that data added to the streams is serial. The outputs of a given invocation of a geometry shader are independent of other invocations (though ordering is respected). A geometry shader generating triangle strips will start a new strip on every invocation.

When a geometry shader output is identified as a System Interpreted Value (e.g. SV\_RenderTargetArrayIndex or SV\_Position), hardware looks at this data and performs some behavior dependent on the value, in addition to being able to pass the data itself to the next shader stage for input. When such data output from the geometry shader has meaning to the hardware on a per-primitive basis (such as SV\_RenderTargetArrayIndex or SV\_ViewportArrayIndex), rather than on a per-vertex basis (such as SV\_ClipDistance[n] or SV\_Position), the per-primitive data is taken from the leading vertex emitted for the primitive.

Partially completed primitives could be generated by the geometry shader if the geometry shader ends and the primitive is incomplete. Incomplete primitives are silently discarded. This is similar to the way the IA treats partially completed primitives.

The geometry shader can perform load and texture sampling operations where screen-space derivatives are not required (samplelevel, samplecmplevelzero, samplegrad).

Algorithms that can be implemented in the geometry shader include:

* Point Sprite Expansion
* Dynamic Particle Systems
* Fur/Fin Generation
* Shadow Volume Generation
* Single Pass Render-to-Cubemap
* Per-Primitive Material Swapping
* Per-Primitive Material Setup - Including generation of barycentric coordinates as primitive data so that a pixel shader can perform custom attribute interpolation (for an example of higher-order normal interpolation, see [CubeMapGS Sample](http://msdn.microsoft.com/en-us/library/windows/desktop/ee416398(v=vs.85).aspx)).

**Rasterizer – Stage**

The main job of the rasterization stage is to compute pixel colors from the projected 3D triangles.

The rasterization stage converts vector information (composed of shapes or primitives) into a raster image (composed of pixels) for the purpose of displaying real-time 3D graphics. During rasterization, each primitive is converted into pixels, while interpolating per-vertex values across each primitive. Rasterization includes clipping vertices to the view frustum, performing a divide by z to provide perspective, mapping primitives to a 2D viewport, and determining how to invoke the pixel shader. While using a pixel shader is optional, the rasterizer stage always performs clipping, a perspective divide to transform the points into homogeneous space, and maps the vertices to the viewport.

Vertices (x,y,z,w), coming into the rasterizer stage are assumed to be in homogeneous clip-space. In this coordinate space the X axis points right, Y points up and Z points away from camera.

You may disable rasterization by telling the pipeline there is no pixel shader (set the pixel shader stage to **NULL** with[**ID3D11DeviceContext::PSSetShader**](http://msdn.microsoft.com/en-us/library/windows/desktop/ff476472(v=vs.85).aspx)), and disabling depth and stencil testing (set **DepthEnable** and **StencilEnable** to **FALSE** in[**D3D11\_DEPTH\_STENCIL\_DESC**](http://msdn.microsoft.com/en-us/library/windows/desktop/ff476110(v=vs.85).aspx)). While disabled, rasterization-related pipeline counters will not update. There is also a complete description of the [rasterization rules](http://msdn.microsoft.com/en-us/library/windows/desktop/cc627092(v=vs.85).aspx).

On hardware that implements hierarchical Z-buffer optimizations, you may enable preloading the z-buffer by setting the pixel shader stage to**NULL** while enabling depth and stencil testing.

##### **Pixel-Shader Stage**

The pixel-shader stage receives interpolated data for a primitive and generates per-pixel data such as color.

Vertex attributes output from the vertex shader (or geometry shader) are interpolated across the triangle in the rasterizer stage. The interpolated values are then fed into the pixel shader as input.

The pixel-shader stage (PS) enables rich shading techniques such as per-pixel lighting and post-processing. A pixel shader is a program that combines constant variables, texture data, interpolated per-vertex values, and other data to produce per-pixel outputs. The rasterizer stage invokes a pixel shader once for each pixel covered by a primitive, however, it is possible to specify a **NULL** shader to avoid running a shader.

When multisampling a texture, a pixel shader is invoked once per-covered pixel while a depth/stencil test occurs for each covered multisample. Samples that pass the depth/stencil test are updated with the pixel shader output color.

The pixel shader intrinsic functions produce or use derivatives of quantities with respect to screen space x and y. The most common use for derivatives is to compute level-of-detail calculations for texture sampling and in the case of anisotropic filtering, selecting samples along the axis of anisotropy. Typically, a hardware implementation runs a pixel shader on multiple pixels (for example a 2x2 grid) simultaneously, so that derivatives of quantities computed in the pixel shader can be reasonably approximated as deltas of the values at the same point of execution in adjacent pixels.

###### Inputs

When the pipeline is configured without a geometry shader, a pixel shader is limited to 16, 32-bit, 4-component inputs. Otherwise, a pixel shader can take up to 32, 32-bit, 4-component inputs.

Pixel shader input data includes vertex attributes (that can interpolated with or without perspective correction) or can be treated as per-primitive constants. Pixel shader inputs are interpolated from the vertex attributes of the primitive being rasterized, based on the interpolation mode declared. If a primitive gets clipped before rasterization, the interpolation mode is honored during the clipping process as well.

Vertex attributes are interpolated (or evaluated) at pixel shader center locations. Pixel shader attribute interpolation modes are declared in an input register declaration, on a per-element basis in either an [argument](http://msdn.microsoft.com/en-us/library/windows/desktop/bb509606(v=vs.85).aspx) or an [input structure](http://msdn.microsoft.com/en-us/library/windows/desktop/bb509668(v=vs.85).aspx). Attributes can be intepolated linearly, or with[centroid sampling](http://msdn.microsoft.com/en-us/library/windows/desktop/ee415231(v=vs.85).aspx). Centroid evaluation is relevant only during multisampling to cover cases where a pixel is covered by a primitive but a pixel center may not be; centroid evaluation occurs as close as possible to the (non-covered) pixel center.

Inputs may also be declared with a [system-value semantic](http://msdn.microsoft.com/en-us/library/windows/desktop/bb509647(v=vs.85).aspx), which marks a parameter that is consumed by other pipeline stages. For instance, a pixel position should be marked with the SV\_Position semantic. The IA stage can produce one scalar for a pixel shader (using SV\_PrimitiveID); the rasterizer stage can also generate one scalar for a pixel shader (using SV\_IsFrontFace).

###### Outputs

A pixel shader can output up to 8, 32-bit, 4-component colors, or no color if the pixel is discarded. Pixel shader output register components must be declared before they can be used; each register is allowed a distinct output-write mask.

Use the depth-write-enable state (in the output-merger stage) to control whether depth data gets written to a depth buffer (or use the discard instruction to discard data for that pixel). A pixel shader can also output an optional 32-bit, 1-component, floating-point, depth value for depth testing (using the SV\_Depth semantic). The depth value is output in the oDepth register, and replaces the interpolated depth value for depth testing (assuming depth testing is enabled). There is no way to dynamically change between using fixed-function depth or shader oDepth.

A pixel shader cannot output a stencil value.

### The Output Merger Stage

After pixel fragments have been generated by the pixel shader, they move to the output merger (OM) stage of the rendering pipeline. In this stage, some pixel fragments may be rejected (e.g., from the depth or stencil buffer tests). Pixel fragments that are not rejected are written to the back buffer. Blending is also done in this stage, where a pixel may be blended with the pixel currently on the back buffer instead of overriding it completely. Some special effects like transparency are implemented with blending; [Chapter 8](mk:@MSITStore:F:\DEVELOPMENT\GAME%20DEVELOPMENT\Engine\Books\Frank%20D.Luna%20-%20Introduction%20to%203D%20Game%20Programming%20with%20DirectX%2010.chm::/93.html) is devoted to blending.

- проекционная матрица задаёт видовой объём и проектирует точки на проекционное окно данного видового объёма

- далее требуется нормализация координат спроектированных точек таким образом, чтобы они не зависели от внешних величин (имеются в виду: r = width/height – коэф. соотношения сторон (aspect ratio) и α - угол вертикального поля видимости, которые задаются пользователем при вызове функции создания проекционной матрицы) и находились в диапазонах значений:

-1≤ x’≤1; -1≤ y’≤1; 0≤ z’≤1 ;

это лишает программиста необходимости всякий раз перед отрисовкой сцены передавать оборудованию значения r и α

- после такой нормализации точки сцены находятся в NDC пространстве, имеющем размеры:

width = height = 2 и depth = 1 (см. диапазоны x’, y’ и z’ выше)

- после умножения точки на проекционную матрицу в координате w этой точки будет сохранена её же координата z, не изменённая проекционным преобразованием; это нужно для последующего перевода точки в NDC-пространство путём деления её координат на сохранённую в w координату z

## 6. Освещение

### 6.1 Глобальное и локальное освещение, материалы

*Глобальное освещение* – модель освещения, которая для расчёта освещения объектов учитывает помимо света, напрямую излучаемого источниками света, также и свет, отражённый от других объектов сцены

Применение моделей глобального освещения позволяет добиться почти фотореалистического качества изображения

*Локальное освещение* – модель освещения, которая учитывает только прямое излучение источников света, пренебрегая светом, отражённым от других объектов. Т.о. каждый объект сцены освещается отдельно от других

- Материалы:

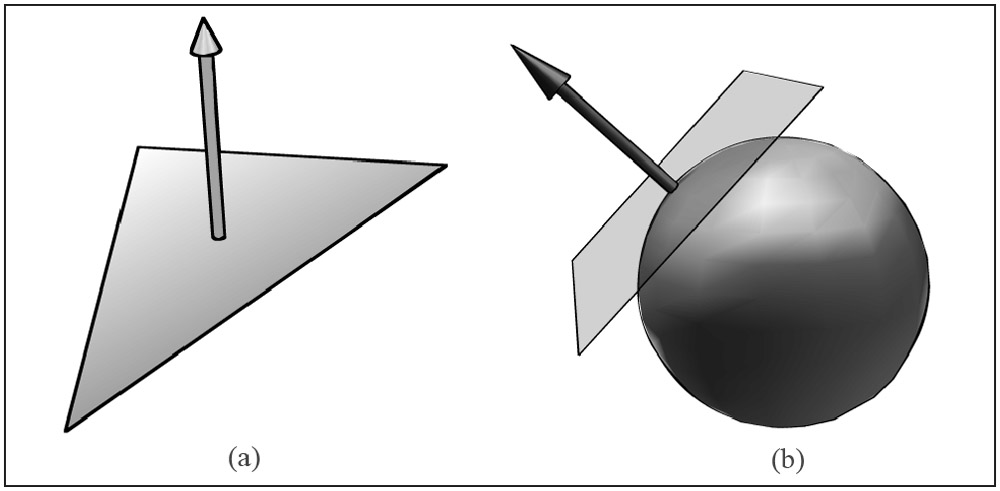
для задания характера взаимодействия света с объектами сцены используются материалы – свойства поверхности каждого объекта – являющиеся моделями материалов, из которых состоят реальные объекты

Материал определяет сколько света поверхность отражает, поглащает и пропускает сквозь себя.

### 6.2 Векторы нормали

Полигональная нормаль (face normal) – нормаль треугольника.

Нормаль поверхности (surface normal) – нормаль касательной плоскости в точке поверхности.

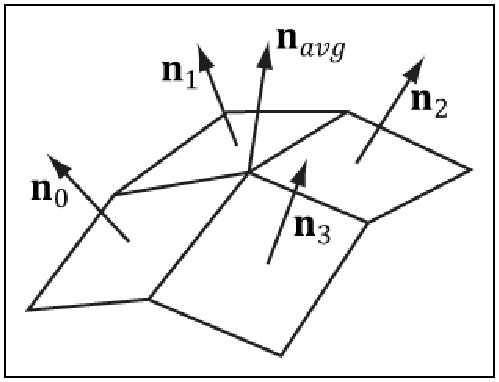


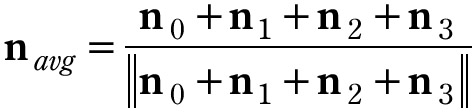
- Для расчёта освещения мы должны найти нормаль поверхности (surface normal) в каждой точке поверхности треугольного сеточного объекта. Таким образом мы сможем определить угол, под которым свет падает на точку поверхности объекта, а значит и интенсивность освещения данной точки.

- Для получения нормалей поверхности во всех точках поверхности объекта нужно задать нормали поверхности только лишь в точках вершин объекта – так называемые вершинные нормали (vertex normals). И уже на стадии растеризации эти вершинные нормали интерполируются по треугольникам, таким образом получаются аппроксимированные нормали поверхности для каждой точки поверхности треугольного сеточного объекта.

- Техника, которая обычно применяется для нахождения нормалей в каждой вершине поверхности треугольного сеточного объекта, называется усреднение вершинных нормалей (vertex normal averaging). Данная методика применяется в случае с недифференцируемой поверхностью. Вершинная нормаль **n** для произвольной вешины **v** сеточного объекта находится путём усреднения полигональных нормалей (face normal) всех треугольников, которые cj вершину **v**.

- В случае с моделированием дифференцируемых поверхностей мы можем получить нормали для каждой точки поверхности при помощи дифференциального исчисления и, соответственно, поскольку вершины сеточного представления поверхности также являются её точками, то нормали в этих вершинах тоже находятся при помощи дифференцирования (использование усреднения вершинных нормалей в таком случае нецелесообразно).





### Закон косинусов Ламберта

– закон, определяющий зависимость интенсивности света от угла его падения на поверхность (controls how much of the original light the surface receives based on the angle between the surface normal and light vector). Аналитически выражается в виде функции:

,

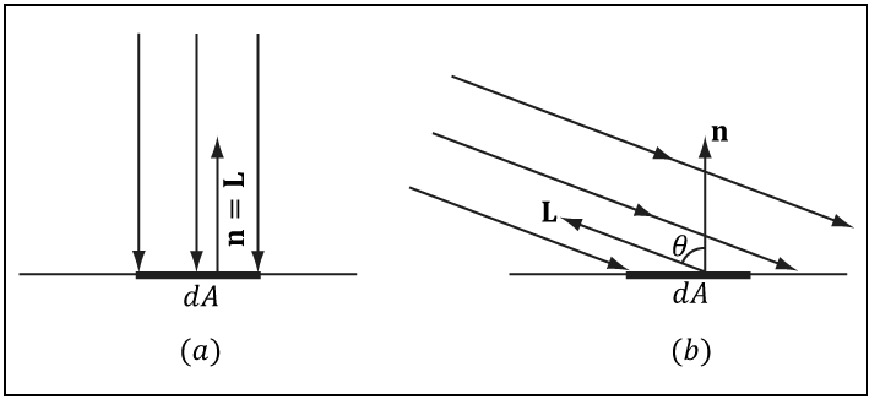
где L – вектор отражённого луча света (направлен от поверхности к источнику света), n – вектор нормали поверхности (surface normal) в точке; оба вектора являются ортами.

Figure 6.8: Consider a small area element dA. (a) The area dA receives the most light when the normal vector n and light vector L are aligned (т.е. L = n). (b) The area dA receives less light as the angle θ between n and L increases (as depicted by the light rays that miss the surface dA).

Т.о. отражённый свет будет иметь максимальную интенсивность при попадании исходящего света в точку поверхности под прямым углом и минимальную – при распространении параллельно поверхности. Функция f(ѳ) возвращает только положительные значения, т.к. берётся максимальное значение из двух – значение косинуса угла тета, если оно не отрицательно, либо ноль, в противном случае. То есть исключается влияние света, излучаемого в диапазоне углов от ±90 до ± 180 градусов к поверхности, т.к. такое возможно в виртуальном мире, но не в физическом (речь о непрозрачных поверхностях).

Вывод:

I(угол падения света) = Закон косинусов Ламберта = f(ѳ) = max(**L** · **n**, 0)

– интенсивность (т.е. яркость, сила) света зависит от угла падения света на поверхность освещаемого объекта и выражается при помощи закона косинусов Ламберта как функция обратного вектора излучения света **L** и вектора **n** - нормали поверхности в точке.

### Модель локального освещения

**В модели вектор L является обратным вектору падающего света, т.к. это позволяет определить угол между нормалью и направлением излучения источника света (закон косинусов Ламберта).**

**Символы элементов уравнения c и m являются сокращениями от английских слов color и material и обозначают, соответственно, цвет, излучаемого света и материал поверхности освещаемого объекта.**

**In our model, a light source emits three different kinds of light:**

* **Ambient light**: to model indirect lighting (reflected off other objects) – аппроксимация глобального освещения. Ambient term of equation:



где **c***A – цвет света, который поверхность получила от источника света косвенно (непрямое освещение, фоновое освещение), т.е. путём отражения от других поверхностей;* **m***A – фоновый материал (фоновая составляющая материала).*

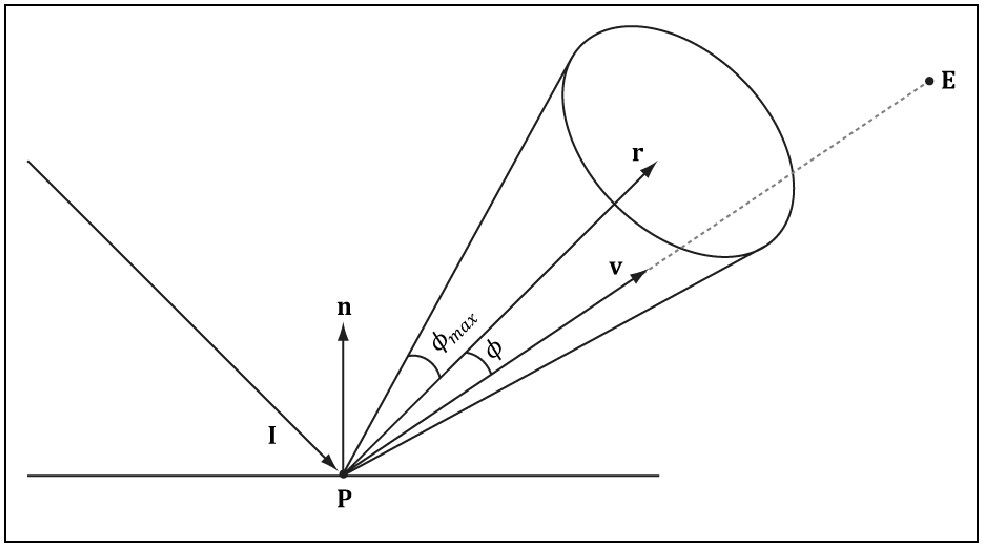
All ambient light does is uniformly brighten up the object a bit — there is no real physics calculation at all. The idea is that the indirect light has scattered and bounced around the scene so many times that it strikes the object equally in every direction.

* **Diffuse light**: to model the direct lighting of relatively rough surfaces (point of view independent, since scatters off the surface equally in all directions, as presumed by model). Diffuse term of the lighting equation:

,

где *kD* = max(**L** · **n**, 0) – коэффициент, выражающий закон Ламберта; **c***D - падающий свет,* **m***D – диффузный материал (диффузная составляющая материала).*

* **Specular light**: to model the direct lighting of relatively smooth surfaces. (View point dependent – amount of received specular light varies accordingly to viewer’s position; refelects within a cone of reflectance).

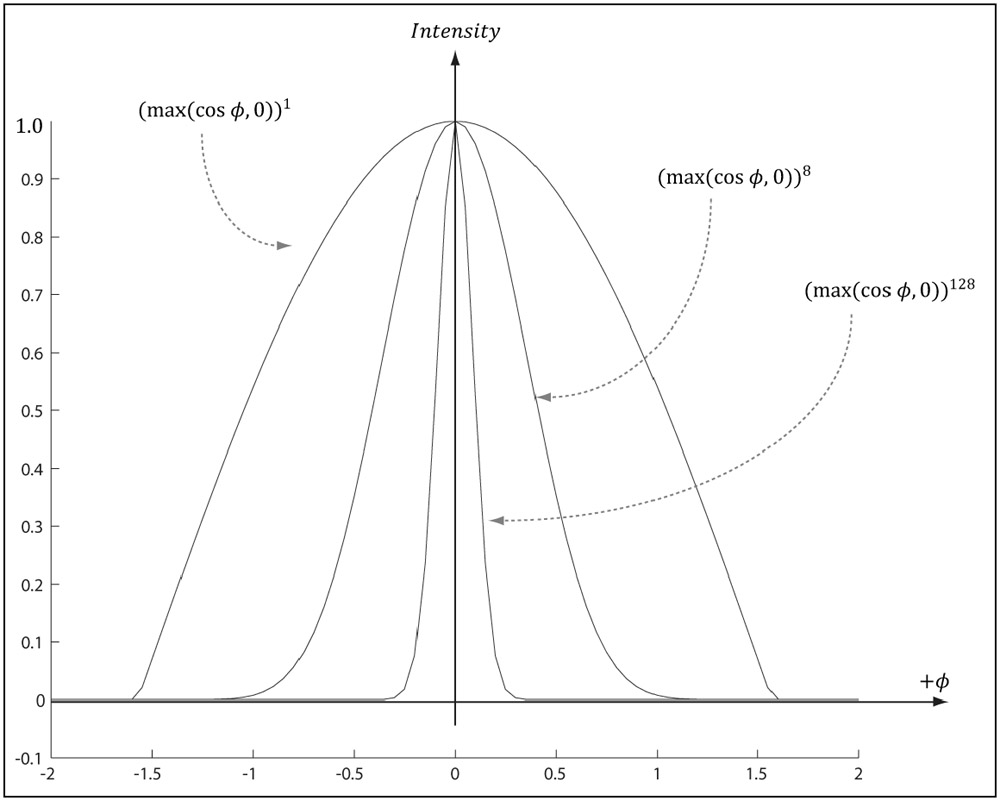


Specular reflections do not scatter in all directions, but instead reflect in a general cone of reflection whose size we can control with a parameter. If v is in the cone, the eye receives specular light; otherwise, it does not. The closer v is aligned with r, the more specular light the eye receives.

r - reflection vector

v – view vector (the unit vector from the surface point **P** to the eye position **E**)

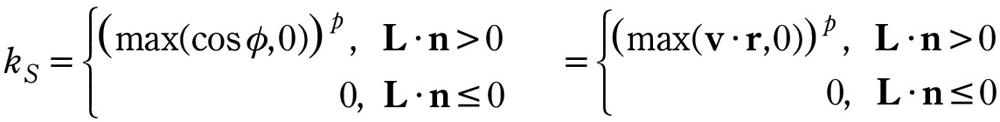
Для задания размеров конуса отражения, т.е. значения угла , в закон косинусов Ламберта вводится степенной показатель *p*. Т.о. зависимость размеров конуса отражения от показателя *p* приобретает обратнопропорциональный характер:



Plots of the cosine functions with different powers of p ≥ 1.

Specular term of equation:

,

где

**c***S – количество зеркального света, излучаемого источником;*

**m***S – зеркальный материал;*

**p –** зеркальная экспонента.

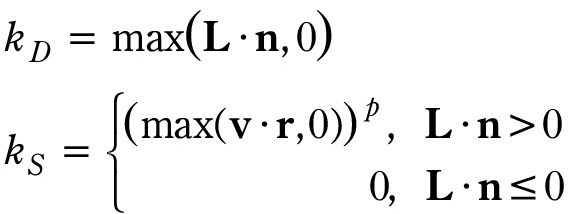
**Correspondingly, a surface point has the following material properties associated with it:**

* Ambient material: the amount of ambient light the surface reflects and absorbs.
* Diffuse material: the amount of diffuse light the surface reflects and absorbs.
* Specular material: the amount of specular light the surface reflects and absorbs.
* Specular exponent: an exponent used in the specular lighting calculation, which controls the cone of reflectance and thus how shiny the surface is. The smaller the cone, the smoother/shinier the surface.

Полностью данная модель локального освещения выглядит так:

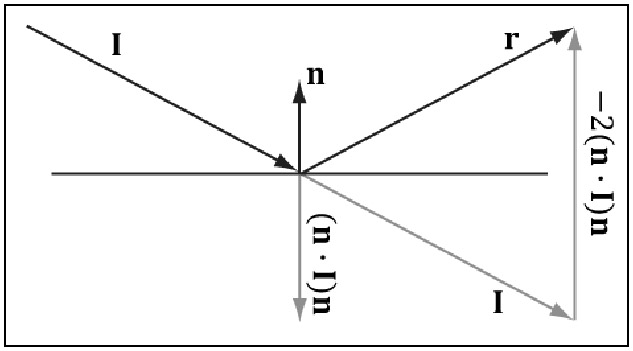
,

где



L = -I

r = **I** − 2(**n** · **I**)**n**



Для расчёта вектора отражения r можно также использовать внутреннюю HLSL функцию reflect.

**?:** Почему во всех формулах для расчёта яркости освещения по закону Ламберта вектор светового луча L является инвертированным вектором направления, в котором распространется поток от источника света? Почему нельзя взять в качестве вектора L вектор направления распространения лучей света от источника к поверхности? Если не брать L = - I, а полагать L равным I, то угол, который будет составлять вектор L с векторм нормали в освещаемой точке будет по абсолютной величине больше на 90 градусов.

The reason for breaking lighting up into three components like this is for the flexibility; an artist has several degrees of freedom to tweak to obtain the desired output. [Figure 6.15](mk:@MSITStore:F:\DEVELOPMENT\GAME%20DEVELOPMENT\Engine\DirectX,%20C++\Books\Introduction%20to%203D%20Game%20Programming%20with%20DirectX%2010.chm::/71.html#ch06fig15F66A748A-0A78-4EB4-B03E-98791450DDBC) shows how these three components work together.

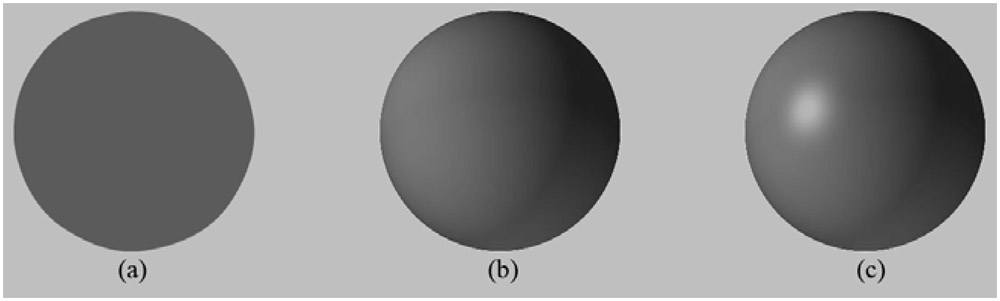


Figure 6.15: (a) Sphere colored with ambient light only, which uniformly brightens it. (b) Ambient and diffuse lighting combined. There is now a smooth transition from bright to dark due to Lambert’s cosine law. (c) Ambient, diffuse, and specular lighting. The specular lighting yields a specular highlight.

We define material values at the vertex level. These values are then linearly interpolated across the 3D triangle to obtain material values at each surface point of the triangle mesh. Our vertex structure looks like this:

struct Vertex

{

D3DXVECTOR3 pos;

D3DXVECTOR3 normal;

D3DXCOLOR diffuse;

D3DXCOLOR spec; // (r, g, b, specPower);

};

D3D10\_INPUT\_ELEMENT\_DESC vertexDesc[] =

{

{"POSITION", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0,

0, D3D10\_INPUT\_PER\_VERTEX\_DATA, 0},

{"NORMAL", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0,

12, D3D10\_INPUT\_PER\_VERTEX\_DATA, 0},

{"DIFFUSE", 0, DXGI\_FORMAT\_R32G32B32A32\_FLOAT, 0,

24, D3D10\_INPUT\_PER\_VERTEX\_DATA, 0},

{"SPECULAR", 0, DXGI\_FORMAT\_R32G32B32A32\_FLOAT, 0,

40, D3D10\_INPUT\_PER\_VERTEX\_DATA, 0}

};

In our implementation, we require that **m***A* = **m***D*; for example, if the surface reflects red diffuse light, then it also reflects red ambient light. Some other implementations allow **m***A* to be different from **m***D* for greater flexibility.

Also, notice that we embed the specular power exponent *p* into the fourth component of the specular material color. This is because the alpha component is not needed for lighting, so we might as well use the empty slot to store something useful. The alpha component of the diffuse material will be used too — for alpha blending.

### Типы источников света

#### Параллельный

Характеризуется одним направлением излучения для любой точки освещаемой поверхности, т.е. интенсивность освещения рассчитывается на основе одного вектора обратного направления света L:

I(p) = L dot n = ls.dir dot n

#### Точечный

Излучает свет во всех направлениях равномерно. Требует рассчёта вектора обратного направления света для каждой точки освещаемой поверхности, т.е. I(p) = L(p) dot n =( ls.pos-p) dot n =light\_dir(p) dot n

#### Прожекторный

Излучает свет в заданном конусе освещения, за исключением чего является точечным источником света. Получение конуса освещения достигается введением коэффициента, подобно тому, как это делается в рассчёте зеркального освещения:

I(p) = ( L(p) dot n )\*k =( ( ls.pos-p) dot n )\*k =(light\_dir(p) dot n)\*k,

где k есть функция от общего направления ИС и направления света в точке p: k = (ls.dir dot light\_dir(p))^t , степенной показатель t варьирует величину конуса (обратнопропорционально).

## 6.12.2 Padding [Structure Packing]

Pad – дополнять, заполнять, раздувать (complement, complete, fill, inflate)

Padding - дополнение

The preceding effect file has a constant buffer with a Light instance. We would like to be able to set this value with one function call. Therefore, in the C++ code we define a structure very similar to the HLSL Light structure:

struct Light

{

Light()

{

ZeroMemory(this, sizeof(Light));

}

D3DXVECTOR3 pos;

float pad1; // not used

D3DXVECTOR3 dir;

float pad2; // not used

D3DXCOLOR ambient;

D3DXCOLOR diffuse;

D3DXCOLOR specular;

D3DXVECTOR3 att;

float spotPow;

float range;

};

The issue with the “pad” variables is to make the C++ structure match the HLSL structure. In the HLSL, structure padding occurs so that elements are packed into 4D vectors, with the restriction that a single element cannot be split across two 4D vectors. Consider the following example:

struct S

{

float3 pos;

float3 dir;

};

If we have to pack the data into 4D vectors, you might think it is done like this:

vector 1: (pos.x, pos.y, pos.z, dir.x)

vector 2: (dir.y, dir.z, empty, empty)

However, this splits the element dir across two 4D vectors, which is not allowed — an element is not allowed to straddle a 4D vector boundary. Therefore, it has to be packed like this:

vector 1: (pos.x, pos.y, pos.z, empty)

vector 2: (dir.x, dir.y, dir.z, empty)

Thus, the “pad” variables in our C++ structure are able to correspond to those empty slots in the padded HLSL structure (since C++ does not follow the same packing rules as HLSL).

If we have a structure like this:

struct S

{

float3 v;

float s;

float2 p;

float3 q;

};

The structure would be padded and the data will be packed into three 4D vectors like so:

vector 1: (v.x, v.y, v.z, s)

vector 2: (p.x, p.y, empty, empty)

vector 3: (q.x, q.y, q.z, empty)

And a final example, the structure:

struct S

{

float2 u;

float2 v;

float a0;

float a1;

float a2;

};

would be padded and packed like so:

vector 1: (u.x, u.y, v.x, v.y)

vector 2: (a0, a1, a2, empty)

- отдельно стоящий float будет использован для дополнения, если предыдущий тип:

- всё дополняется до 4D-вектора

- вектор не может быть распределён между двумя 4D-векторами

## Текстурные массивы

- Texture array is represented by the **ID3D10Texture2D** interface (the same one used for single textures)

- In an effect file, a texture array is represented by the Texture2DArray type

- When using a texture array, three texture coordinates are required: uvw, где w – индекс текстуры в массиве

- One of the advantages of using texture arrays is that we are able to draw a collection of primitives, with different textures, in one draw call

- ID примитива будет служить индексом его текстуры в текстурном массиве

Процесс создания текстурного масива:

1. Create each texture from file individually one-by-one.
2. Create the texture array.
3. Copy each individual texture into the elements of the texture array.
4. Create a shader resource view to the texture array.

## Кубические текстуры [Cube maps]

Кубическая текстура ­- текстура из 6 текстур, которые можно разместить по сторонам куба, центрированного и сонаправленного относительно какой-либо СК, физически представляет собой текстурный массив из 6 элементов

- A cube map can be represented by the **ID3D10Texture2D** interface **as a texture array** ([§10.3](mk:@MSITStore:F:\DEVELOPMENT\GAME%20DEVELOPMENT\Engine\DirectX,%20C++\Books\Introduction%20to%203D%20Game%20Programming%20with%20DirectX%2010.chm::/115.html#718))

- In the HLSL, a cube map is represented by the TextureCube type. Sampling of a cube map:

TextureCube gCubeMap;

SamplerState gTriLinearSam

{

Filter = MIN\_MAG\_MIP\_LINEAR;

AddressU = Wrap;

AddressV = Wrap;

};

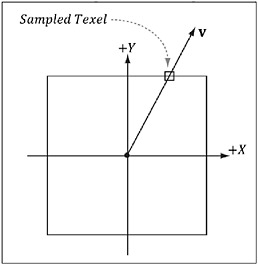
...

// in pixel shader

float3 v = float3(x,y,z); // some lookup vector

float4 color = gCubeMap.Sample(gTriLinearSam, v);

- To identify a texel in a cube map, we use 3D texture coordinates, which define a 3D *lookup* vector **v** originating at the center of the cube map. The texel of the cube map that **v** intersects is the texel corresponding to the 3D coordinates of **v**.



The illustration is in 2D for simplicity; in 3D the square becomes a cube. The square denotes a cube map centered and axis-aligned with some coordinate system. We shoot a vector v from the origin. The texel v intersects is the sampled texel. In this illustration, v intersects the cube face corresponding to the +Y-axis.

The magnitude of the lookup vector is unimportant; only the direction matters. Two vectors with the same direction but different magnitudes will sample the same point in the cube map.

- Каждая грань КТ опред-ся нормалью грани (010), (100), (±*X*, ±*Y*, ±*Z*).

For the purposes of identifying a cube map face, Direct3D provides the D3D10\_TEXTURECUBE\_FACE enumerated type:

typedef enum D3D10\_TEXTURECUBE\_FACE

{

D3D10\_TEXTURECUBE\_FACE\_POSITIVE\_X = 0,

D3D10\_TEXTURECUBE\_FACE\_NEGATIVE\_X = 1,

D3D10\_TEXTURECUBE\_FACE\_POSITIVE\_Y = 2,

D3D10\_TEXTURECUBE\_FACE\_NEGATIVE\_Y = 3,

D3D10\_TEXTURECUBE\_FACE\_POSITIVE\_Z = 4,

D3D10\_TEXTURECUBE\_FACE\_NEGATIVE\_Z = 5,

} D3D10\_TEXTURECUBE\_FACE;

A cube map is stored in a texture array with six elements:

* Index 0 refers to the +X face.
* Index 1 refers to the −X face.
* Index 2 refers to the +Y face.
* Index 3 refers to the −Y face.
* Index 4 refers to the +Z face.
* Index 5 refers to the −Z face.

- Основное применение: текстуры окружения (environment mapping) - текстурирование небосвода, создание фона окружения; построение отражений криволинейных поверхностей

- Даёт значительные погрешности при моделировании отражений плоских поверхностей из-за того, что в основе метода лежит использование для определения проецируемого текселя векторов, которые не хранят положение точки проецирования. Так для двух разных точек проецирования могут существовать два одинаковых вектора отражения, что приведёт к проецированию двух одинаковых текселей кубической текстуры

- Для приближённого решения проблемы с плоскими пов-ми см. Brennan, Chris. “Accurate Reflections and Refractions by Adjusting for Object Distance,” Direct3D ShaderX: Vertex and Pixel Shader Tips and Tricks. Wordware Publishing, Inc., 2002.

- An environment map captures the surrounding environment about a point with six images. These images can then be stored in a cube map. With environment maps we can easily texture distant background, sky or approximate reflections. In other words an *environment map* is a cube map where the cube faces store the surrounding images of an environment.



- Cube maps can be made from six individual images using the DirectX Texture Tool. Cube maps can then be saved to file with the DDS image format. Because cube maps store six 2D textures, which can consume a lot of memory, a compressed DDS format should be used. The **D3DX10CreateTextureFromFile** function can **load** a DDS file that stores a cube map **into** an **ID3D10Texture2D** object **as** a **texture array**.

**Пример создания кубической текстуры:**

D3DX10\_IMAGE\_LOAD\_INFO loadInfo;

loadInfo.MiscFlags = D3D10\_RESOURCE\_MISC\_TEXTURECUBE;

ID3D10Texture2D\* tex = 0;

HR(D3DX10CreateTextureFromFile(md3dDevice, filename.c\_str(),

&loadInfo, 0, (ID3D10Resource\*\*)&tex, 0) );

D3D10\_TEXTURE2D\_DESC texDesc;

tex->GetDesc(&texDesc);

D3D10\_SHADER\_RESOURCE\_VIEW\_DESC viewDesc;

viewDesc.Format = texDesc.Format;

viewDesc.ViewDimension = D3D10\_SRV\_DIMENSION\_TEXTURECUBE;

viewDesc.TextureCube.MipLevels = texDesc.MipLevels;

viewDesc.TextureCube.MostDetailedMip = 0;

ID3D10ShaderResourceView\* rv = 0;

HR(md3dDevice->CreateShaderResourceView(tex, &viewDesc, &rv));

## Динамические кубические текстуры [Dynamic cube maps]

Динамическая кубическая текстура – это КТ, обновляющаяся для учёта изменений сцены.

- Примером случая, когда необходимо применение ДКТ может служить построение отражения на поверхности объекта, который перемещается либо вращается в пространстве, либо относительно него просиходит изменение объектов сцены.

- В данном случае применение простого кубического текстурирования приведёт к неадекватному построению отражения, во первых в нём не будет присутствовать ничего кроме того, что запечатлено в текстуре, т.е. окружающие объекты будут отсутствовать; во вторых отражение не будет учитывать изменений во взаимном расположении объектов и будет таким как если бы этих изменений не было, т.е. из КТ для построения отражения будут браться тексели, не соответствующие состоянию текстурируемого объекта. Для разных состояний объекта брались бы одинакове тексели. Это связано с уже упомянутым выше недостатком кубического текстурирования, который состоит в том, что при текстурировании для определения нужного текселя применется поисковой вектор, не учитывающий положения объекта, а не луч, который хранит в себе информацию не только об ориентации наблюдателя, но и о положении объекта. Т.о. при разных положениях объекта, но при одной и той же ориентации наблюдателя из КТ будет браться один и тот же тексель.

- ДКТ будет обновляться после изменения пространственных характеристик объекта, на который она налаживается, либо объектов его окружающих, и соответственно учитывать изменения сцены.

- На современном этапе развития графических технологий ДКТ получается путём рендеринга сцены одновременно в каждую из шести граней кубической текстуры каждый раз когда строится кадр, т.о. ДКТ создаётся за один проход. Для этого способа применяются такие, впервые введённые в Direct3D 10, средства как геометрический шейдер и массив целей вывода (render target array). Второй способ получения ДКТ является гораздо менее эффективным – это поочерёдный рендеринг сцены в каждую грань текстуры за шесть проходов, что существенно увеличивает расходы производительности. Данный подход использовался в Direct3D 9 и в более ранних версиях API.

- Для одновременного рендеринга сразу во все 6 граней кубической текстуры используется массив целей вывода из шести целей и геометрический шейдер, который позволяет производить вывод треугольника на заданную цель вывода (render target) в массиве целей вывода.

- A render target array allows multiple render target and depth stencil textures to be active simultaneously. By using an array of six render targets, one for each face of the cube texture, all six faces of the cube can be rendered together.

- When the geometry shader emits a triangle, it can control which render target in the array the triangle gets rasterized on.

- For every triangle that gets passed to the geometry shader, six triangles are generated in the shader and output to the pixel shader, one triangle for each render target.

- For each output primitive, the geometry shader can also control to which element slice of the render target array the primitive gets emitted.

- The render target array in Direct3D 10 is a feature that enables an application to render onto multiple render targets simultaneously at the primitive level. The application uses a geometry shader to output a primitive and select which render target in the array should receive the primitive.

**Пример построения динамической кубической текстуры:**

Каждый кадр рендерится в *динамическую кубическую текстуру* - *массив целей вывода* (render target array) из 6 целей (render targets) каждая для одной грани *кубической текстуры*. Для этого создаётся *вид цели вывода* (render target view), который представляет собой массив из 6 целей вывода, и перед тем как осуществить отрисовку (рендеринг) сцены в кубическую текстуру данный вид цели вывода делается текущим видом для рендеринга, таким образом все 6 граней кубической текстуры одновременно растеризируются. Это является единственным отличием от процесса создания КТ, для которой не создаётся вида цели вывода, т.к. она загружается из файла, а не получается путём рендеринга в неё сцены. Далее (после рендеринга в текстуру) связанный с ДКТ *шейдерный вид ресурса* (shader resource view) может быть использован в качестве вида обычной КТ для передачи её в шейдер, так же как это было и со статической КТ, загружаемой из файла.

Помимо самой кубической текстуры ID3D10Texture2D, которая будет использована как холст для ДКТ, параллельно должны быть и другие ресурсы, вовлечённые в процесс создания ДКТ, это: кубическая текстура глубины и трафарета (cubic depth stencil texture, ID3D10Texture2D), вид глубины и трафарета для кубической текстуры (depth stencil view, ID3D10DepthStencilView), упомянутый выше вид цели вывода для КТ как массив целей вывода из 6 элементов для 6 граней КТ (render target view, ID3D10RenderTargetView), шейдерный вид ресурса для КТ (ID3D10ShaderResourceView).

Общий алгоритм:

1. **Кубическая текстура глубины и трафарета. Обычная КТ ID3D10Texture2D**, созданная с флагом D3D10\_TEXTURE2D\_DESC::BindFlags = **D3D10\_BIND\_DEPTH\_STENCIL**; Параметры D3D10\_TEXTURE2D\_DESC::Width и D3D10\_TEXTURE2D\_DESC::Height устанавливаются в соответствие с размерами; параметр D3D10\_TEXTURE2D\_DESC::ArraySize = **6** – массив из шести элементов по числу граней КТ:

// Create cubic depth stencil texture.

D3D10\_TEXTURE2D\_DESC dstex;

dstex.Width = 1024;

dstex.Height = 1024;

dstex.MipLevels = 1;

dstex.ArraySize = 6;

dstex.SampleDesc.Count = 1;

dstex.SampleDesc.Quality = 0;

dstex.Format = DXGI\_FORMAT\_D32\_FLOAT;

dstex.Usage = D3D10\_USAGE\_DEFAULT;

dstex.BindFlags = D3D10\_BIND\_DEPTH\_STENCIL;

dstex.CPUAccessFlags = 0;

dstex.MiscFlags = D3D10\_RESOURCE\_MISC\_TEXTURECUBE;

HR( md3dDevice->CreateTexture2D( &dstex, NULL, &cubicDStex ) );

1. **Вид глубины и трафарета для кубической текстуры из предыдущего пункта. ID3D10DepthStencilView.** D3D10\_DEPTH\_STENCIL\_VIEW\_DESC::ViewDimension = **D3D10\_DSV\_DIMENSION\_TEXTURE2DARRAY** – указвает на то что это будет текстурный массив; D3D10\_DEPTH\_STENCIL\_VIEW\_DESC::Texture2Darray::FirstArraySlice = **0** и D3D10\_DEPTH\_STENCIL\_VIEW\_DESC::Texture2Darray::ArraySize = **6** – задают длину массива равной шести элементам:

// Create the depth stencil view for the entire cube

D3D10\_DEPTH\_STENCIL\_VIEW\_DESC DescDS;

DescDS.Format = DXGI\_FORMAT\_D32\_FLOAT;

DescDS.ViewDimension = D3D10\_DSV\_DIMENSION\_TEXTURE2DARRAY;

DescDS.Texture2DArray.FirstArraySlice = 0;

DescDS.Texture2DArray.ArraySize = 6;

DescDS.Texture2DArray.MipSlice = 0;

HR( md3dDevice->CreateDepthStencilView( cubicDStex, &DescDS, &dynamicCubeMapDSV ) );

1. **Создание непосредственно самой кубической текстуры ID3D10Texture2D.** D3D10\_TEXTURE2D\_DESC::BindFlags = **D3D10\_BIND\_RENDER\_TARGET | D3D10\_BIND\_SHADER\_RESOURCE**; D3D10\_TEXTURE2D\_DESC::MiscFlags = **D3D10\_RESOURCE\_MISC\_GENERATE\_MIPS | D3D10\_RESOURCE\_MISC\_TEXTURECUBE**;

// Create the cube map for env map render target

dstex.Format = DXGI\_FORMAT\_R8G8B8A8\_UNORM;

dstex.BindFlags = D3D10\_BIND\_RENDER\_TARGET | D3D10\_BIND\_SHADER\_RESOURCE;

dstex.MiscFlags = D3D10\_RESOURCE\_MISC\_GENERATE\_MIPS | D3D10\_RESOURCE\_MISC\_TEXTURECUBE;

dstex.MipLevels = 9;

HR( md3dDevice->CreateTexture2D( &dstex, NULL, &dynamicCubeMap ) );

1. **Вид цели вывода для КТ как массив целей вывода из 6 элементов для 6 граней КТ.** **ID3D10RenderTargetView.** D3D10\_RENDER\_TARGET\_VIEW\_DESC::ViewDimension = **D3D10\_DSV\_DIMENSION\_TEXTURE2DARRAY** – указвает на то что это будет текстурный массив; D3D10\_RENDER\_TARGET\_VIEW\_DESC::Texture2Darray::FirstArraySlice = **0** и D3D10\_RENDER\_TARGET\_VIEW\_DESC::Texture2Darray::ArraySize = **6** – задают длину массива равной шести элементам:

// Create the 6-face render target view

D3D10\_RENDER\_TARGET\_VIEW\_DESC DescRT;

DescRT.Format = dstex.Format;

DescRT.ViewDimension = D3D10\_RTV\_DIMENSION\_TEXTURE2DARRAY;

DescRT.Texture2DArray.FirstArraySlice = 0;

DescRT.Texture2DArray.ArraySize = 6;

DescRT.Texture2DArray.MipSlice = 0;

HR( md3dDevice->CreateRenderTargetView( dynamicCubeMap, &DescRT, &dynamicCubeMapRTV ) );

1. **Шейдерный вид ресурса для КТ. ID3D10ShaderResourceView.** D3D10\_SHADER\_RESOURCE\_VIEW\_DESC::ViewDimension = **D3D10\_SRV\_DIMENSION\_TEXTURECUBE** :

// Create the shader resource view for the cubic env map

D3D10\_SHADER\_RESOURCE\_VIEW\_DESC SRVDesc;

ZeroMemory( &SRVDesc, sizeof( SRVDesc ) );

SRVDesc.Format = dstex.Format;

SRVDesc.ViewDimension = D3D10\_SRV\_DIMENSION\_TEXTURECUBE;

SRVDesc.TextureCube.MipLevels = 9;

SRVDesc.TextureCube.MostDetailedMip = 0;

HR(md3dDevice->CreateShaderResourceView( dynamicCubeMap, &SRVDesc, &dynamicCubeMapSRV ));

1. Далее сохраняются текущие DSV, RTV и Viewport.
2. Устанавливается новый Viewport, созданный для ДКТ.
3. Создаётся массив видовых матриц, которыми будет задаваться пространственные положение и ориентация камеры: положение должно соответствовать расположению текстурируемого объекта, а ориентация камеры должна изменяться на 90 градусов по направлению нормалей каждой из шести граней куба.
4. Перед отрисовкой в ДКТ текущим *видом цели вывода* и *видом глубины и трафарета* устанавливаются *массив целей вывода* и DSV кубической текстуры при помощи метода ID3D10Device::OMSetRenderTargets().
5. В шейдер передаётся ранее созданный массив видовых матриц: pfxmViewCM->SetMatrixArray( ( float\* )amViewCM, 0, 6 );
6. Производится первый проход рендера. Сцена рендерится в ДКТ.
7. Затем восстанавливаются изначальные DSV, RTV и Viewport. Производится второй проход рендера, в котором в качестве кубической текстуры используется полученная перед этим ДКТ, т.е в шейдер передаётся её вид:

// установка кубической текстуры

if ( isRenderToCubeMap ) mfxCubeMapVar->SetResource(mEnvMapRV);

else mfxCubeMapVar->SetResource(dynamicCubeMapSRV);

Сцена выводится на экран.

## Нормальное картирование [Normal mapping]

- В данном методе текстурирования используются нормальные текстуры – текстуры, хранящие в каждом текселе нормали в сжатом формате, т.е. вместо RGB каналов там хранятся кординаты x,y,z нормалей.

- Теперь можно значительно улучшить качество освещения благодаря использованию попиксельных нормалей, что позволяет учесть геометрические свойства поверхности, отражённые в текстуре, и более реалистично передать детали её рельефа, в отличие от обычного текстурирования где для расчёта освещения использовались интерполированные нормали вершин треугольников, которые отражали только лишь геометрию поверхности, находящегося под текстурой, сеточного объекта.

- Нормальное текстурирование позволяет строить динамическое освещение поверхностей объектов, учитывающее геометрические свойства этих поверхностей, отражённые в текстуре, например - выпуклости, впадины, трещины, царапины, щели.

- Для генерирования нормальных текстур используются различные программы, которые могут создавать НТ на основе обычных текстур, также НТ могут быть созданы на основе высокополигональных сеточных объектов.

- Нормали, хранящиеся в текстуре задаются своими координатами относительно касательного пространства (tangent space) - СК текстуры, задаваемой базисными векторами T - tangent, B – bitangent / binormal, N – normal ( x или u, y или v, z или w соответственно). Вектор N является нормалью к треугольнику. Касательное пространство получило своё название т.к. представляет собой пространство, плоскость TB которого является касательной и компланарной для треугольника либо плоскость TB может быть касательной для вершины треугольника. Т.о. для каждой модели, состоящей из n полигонов, мы имеем n базисов TBN для всех треугольников или 3n базисов TBN для всех вершин модели.

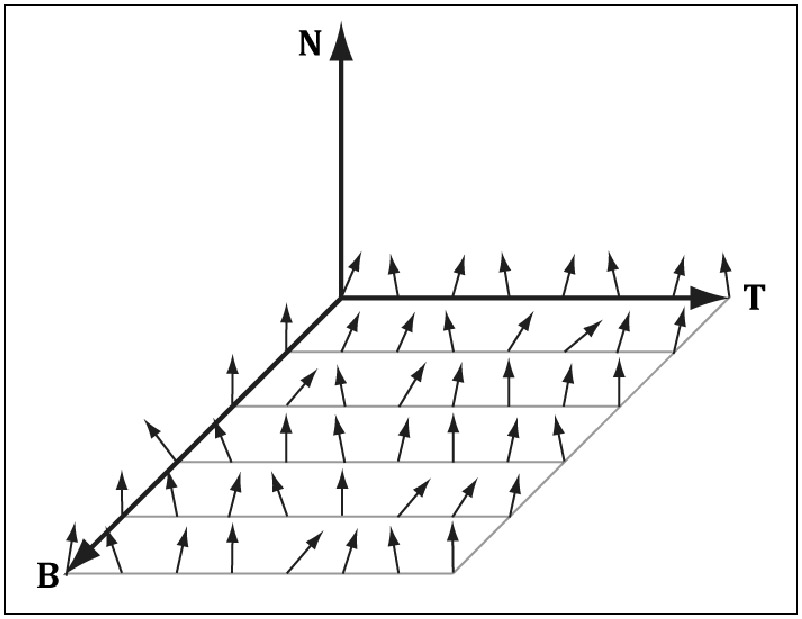
- Для расчёта освещения при помощи текстурных нормалей необходимо найти векторы T, B, N касательного пространства, зная матрицу которого можно осуществлять преобразование объектов, например камеры и источников света из мирового пространства в касательное пространство текстуры, т.к. текстурные нормали и объекты сцены заданы относительно разных систем отсчёта. Если использовать TBN базисы для треугольников, то освещение получится неплавным, триангулированным, т.к. касательное пространство постоянно на всей плоскости треугольника. Т.о нужно нужно задавать касательные векторы повершинно.

- Для наиболее оптимального, с точки зрения вычислительной нагрузки, расчёта освещения нужно применять преобразование источников света и камеры из мирового пространства в касательное, т.к. это потребует преобразовать всего лишь несколько объектов, в то время как расчёт освещения в мировом пространстве потребует совершить преобразование миллионов векторов нормали для каждого пикселя текстуры.

-Касательное пространство расчитывается сперва для каждого полигона (треугольника) модели и затем для каждой вершины треугольника. Методы нахождения T, B, N базиса для каждой вершины модели могут быть разными, одним из самых простых и распространённых является усреднение векторов T, B, N каждого из треугольников, содержащих данную вершину.

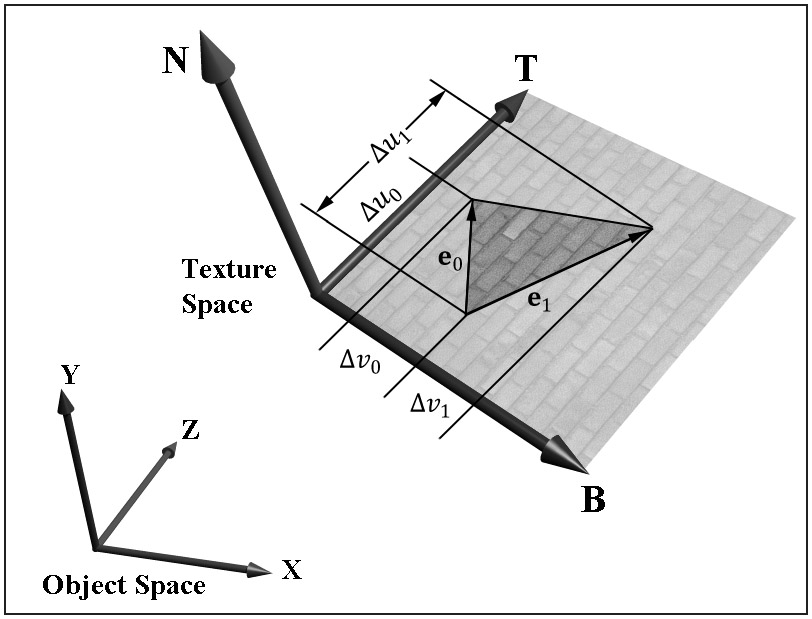
Методика расчёта векторов T, B, N относительно объектного пространства модели, конечная цель - получение матрицы базиса TBN для перевода векторов нормали нормальной текстуры из касательного пространства в мировое пространство:

Figure 12.3: Normals stored in a normal map relative to a texture space coordinate system defined by the vectors T (x-axis), B (y-axis), and N (z-axis). The T vector runs right horizontally to the texture image, the B vector runs down vertically to the texture image, and N is orthogonal to the texture plane.



Now, as [Figure 12.3](mk:@MSITStore:F:\DEVELOPMENT\GAME%20DEVELOPMENT\Engine\DirectX,%20C++\Books\Introduction%20to%203D%20Game%20Programming%20with%20DirectX%2010.chm::/129.html#789) shows, the normal vectors in a normal map are defined relative to the texture space. But our lights are defined in world space. In order to do lighting, the normal vectors and lights need to be in the same space. So our first step is to relate the tangent space coordinate system with the object space coordinate system the triangle vertices are relative to. Once we are in object space, we can use the world matrix to get from object space to world space (the details of this are covered in the next section). Let **v**0, **v**1, and **v**2 define the three vertices of a 3D triangle with corresponding texture coordinates (*u*0, *v*0), (*u*1, *v*1), and (*u*2, *v*2) that define a triangle in the texture plane relative to the texture space axes (i.e., **T** and **B**). Let **e**0 = **v**1 − **v**0 and **e**1 = **v**2 − **v**0 be two edge vectors of the 3D triangle with corresponding texture triangle edge vectors ( Δ*u*0, Δ*v*0) = (*u*1 − *u*0, *v*1 − *v*0) and (Δ*u*1, Δ*v*1) = (*u*2 − *u*0, *v*2 − *v*0).

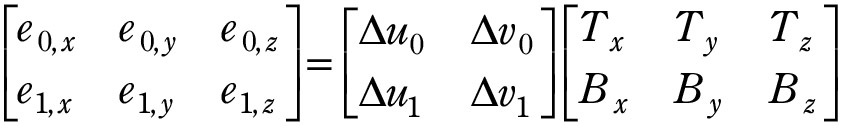
Figure 12.4: The relationship between the texture space of a triangle and the object space.



From [Figure 12.4](mk:@MSITStore:F:\DEVELOPMENT\GAME%20DEVELOPMENT\Engine\DirectX,%20C++\Books\Introduction%20to%203D%20Game%20Programming%20with%20DirectX%2010.chm::/130.html#ch12fig04D6939890-192F-4612-8146-61EB7D3EA455), it is clear that



Representing the vectors with coordinates **relative to object space**, we get the matrix equation:



Note that we know the object space coordinates of the triangle vertices; hence we know the object space coordinates of the edge vectors, so the matrix

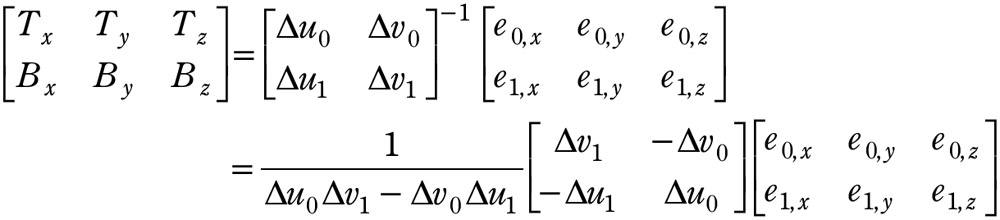


is known.

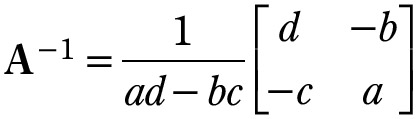
Likewise, we know the texture coordinates, so the matrix



is known. Solving for the **T** and **B** object space coordinates we get:



In the above, we used the fact that the inverse of a matrix  is given by:



Note that the vectors **T** and **B** are generally not unit length in object space, and if there is texture distortion, they will not be orthonormal either.

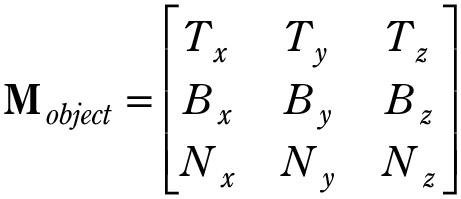
Вектор N является обычной нормалью треугольника и может быть найден как векторное произведение векторов T и B.

Теперь мы имеем касательное пространство для треугольника, используя которое мы получим касательно пространство для вершины модели, для чего необходимо расчитать касательные векторы T и B для вершины, что достигается путём усреднения этих векторов по принципу усреднения векторов номали, которое делалось для аппроксимации плавной поверхности:

1. The tangent vector **T** for an arbitrary vertex **v** in a mesh is found by averaging the tangent vectors of every triangle in the mesh that shares the vertex **v**.
2. The bitangent vector **B** for an arbitrary vertex **v** in a mesh is found by averaging the bitangent vectors of every triangle in the mesh that shares the vertex **v**.

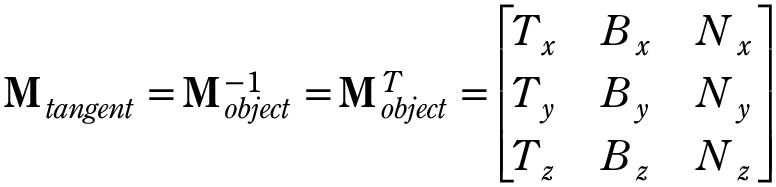
После усреднения базисы TBN должны быть ортонормированны, что обычно достигается при помощи процедуры Грам-Шмидта (Gram-Schmidt procedure).

На данном этапе у нас есть базисы TBN для кждой вершины модели. К тому же координаты векторов TBN заданы относительно объектного пространства модели. Теперь мы можем перевести координаты текстурных нормалей из касательного пространства в объектное пространство при помощи матрицы:



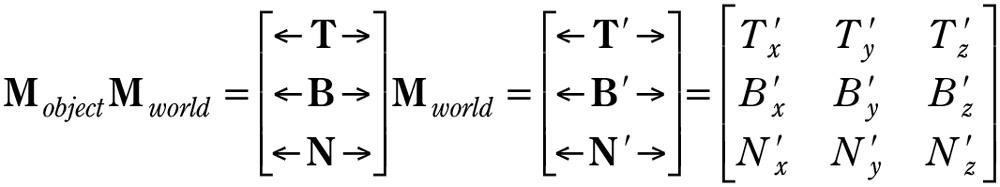
Since this matrix is orthogonal, its inverse is its transpose. Thus, the change of coordinate matrix from object space to tangent space is:

Поскольку матрица является ортогональной, её инверсия будет равна её транспонированию. Таким образом матрица перехода от объектного пространства в касательно пространство равна:



Для перевода нормалей в мировое пространство матрицу **M***object* нужно умножить на матрицу мировой СК **M***world*:





где **T**′ = **T** · **M***world*, **B**′ = **B** · **M***world*, and **N**′ = **N** · **M***world*. Т.о. для перехода из касательного пространства непосредственно в мировое протсранство мы просто должны описать касательныый базис в мировых координатах, что можно сделать путём преобразования TBN базиса из координат объектного пространства к координатам мирового пространства.

Следовательно для перехода из мирового пространства в касательное нужно взять обраную матрицу от произвдения матриц **M***object* и **M***world*.

## Теневое картирование [Shadow mapping]

**I – Создание карты глубин пикселей, расположенных на проекционной плоскости конуса видимости источника света**

**Построение карты глубин точек сцены (ShadowMap или DepthMap):**

- создать 2D-текстуру depthMap и её DSV и SRV; создать VP

- привязать DSV с нулевым RTV к OM и установить VP

- отрисовать сцену в depthMap - build\_depthmap:

- VS:

- преобразование точек сцены к проекционному пространству ИС – матрица LightWVP

- PS:

- исключить из текстур объектов сцены прозрачные пиксели, т.к. они не могут отбрасывать тень

**Использование DepthMap для наложения теней:**

- установить прежние DSV, RTV и VP

- отрисовать сцену, используя технику наложения теней - tex.fx:

- в вершинном шейдере:

- преобразование точек сцены к проекционному пространству ИС путём умножения на матрицу LightWVP

1) В геометрическом шейдере точки сцены преобразуется к видовому пространству (ВП) источника света (ИС). Здесь же сравниваются значения координаты Z точек объектов – z-тест, т.е. глубиной является расстояние вдоль оси Z локального пространства ИС. Т.о. на вход в пиксельный шейдер поступают точки, прошедшие тест глубины;

2) В пиксельном шейдере формируется карта затенения – 2D-текстура, заполненная, соответствующими прошедшим z-тест пикселям, значениями z/w . Отношения z/w есть нормализованные глубины пикселей и следовательно лежат в диапазоне от 0 до 1 ;

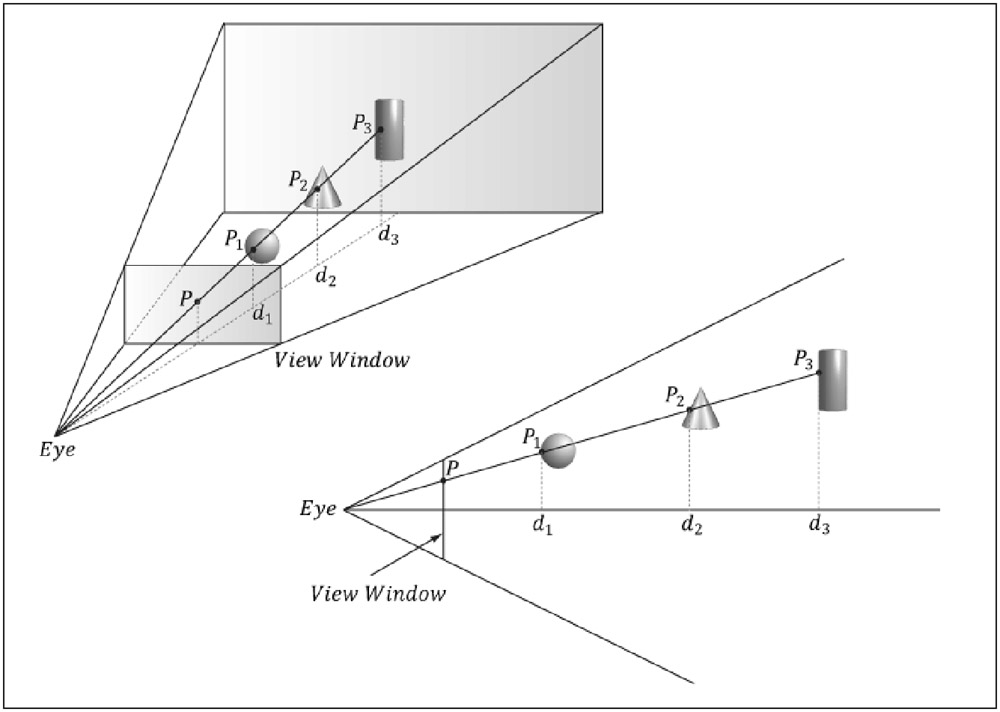
В результате имеем: 2D-карту глубин точек, спроецированных на видовую плоскость базиса ИС; поскольку проводился z-тест, то в текстуре будут содержаться глубины только тех точек сцены, которые видны с позиции базиса ИС, т.е. из всех точек лежащих на одной прямой будет взята ближайшая к ИС, та которая находится на проекционной плоскости базиса ИС.

**II – Отрисовка сцены, наложение теней**

1) Сцена преобразовывается к ВП камеры;

2) Сравниваются расстояние от ИС до точки взятой в объёме с значением в карте глубин, которое соответствует ближайшей к ИС точке на данном луче;

3) Если расстояние в объёме больше, значит точка находится в тени, т.к. лежит дальше проекционной плоскости базиса ИС



Здесь, View Window - проекционная плоскость базиса ИС; Р – точка сцены, значение глубины которой хранится в карте затенения.

В результате имеем: сравниваются расстояние до точки P с расстояниями до точек Pk .

### Артефакаты

Причиной дефектов отображения при теневом картировании является тот факт, что одному значению текселя в карте глубин соответствует сразу площадка пикселей (в силу того, что ограниченное разрешение карты глубин не позволяет сопоставить один тексель одной точке сцены) сцены, следовательно все пиксели такой площадки будут сравниваться с одним и темже текселе карты глубин, т.е. их глубины будут сравниваться с одним значением глубины, хранящимся в текселе. Следовательно все пиксели площадки будут одинаково проходить теневой тест – будут или дальше текселя, или ближе (в тени или на свету, соответственно). Т.о. целые площадки пикселей будут в тени или на свету, отсюда – эффект лестничной кромки.

Можно увеличить разрешение карты глубин. Тогда уменьшится уровень дискретности площадок и одному текселю будет соответствовать меньшая площадка пикселей и уже больше пикселей будут в силу этого проходить теневой тест правильно.