OPEN CASCADE 学习笔记
——标准内存分配器接口

著: Roman Lygin

译：George Feng

这是一篇关于开源三维建模软件 OPEN CASCADE 内核的博文：ROMAN LYGIN 是 OPEN CASCADE 的前程序开发员和项目经理，曾经写过许多关于该开源软件开发包的深入文章，可以在他的博客(HTTP://OPENCASCADE.BLOGSPOT.COM)上面找到这些文章。

序

在 Open Cascade 的论坛上知道了 Roman Lygin 在他的博客上写了 Open Cascade notes 系列文章，考虑到 Open Cascade 的学习资料并不多，于是从他的博客上下载了其中绝大部分文章，将其翻译过来以方便大家学习交流。如果大家发现文中翻译有错误或不足之处，望不吝赐教，可以发到我的邮箱 fenghongkui@sina.com.cn，十分感谢。

2012 年 11 月 22 日星期四

标准内存分配器接口

大家都知道，Open CASCADE 有自己的内存分配机制，其接口是 Standard::Allocate()和 Standard::Free()。它们将分配和释放内存请求提交给内存管理器，可以是 a) 缺省的系统分配器(假如环境变量 MMGT_OPT=0)，也可以是 b) Open CASCADE 自己的分配器(MMGT_OPT=1)，也可以是 c) Intel TBB(MMGT_OPT=2)。

Open CASCADE 的大多数类(例如，所有的类都是在 cdl 文件中定义的)都重新定义了 new 和 delete 操作，其中分别使用了 Standard::Allocate() 和

Standard::Free()——可以看看任意一个 hxx 文件。

使用这些内存分配机制(common memory allocation mechanism)会减少使用内存的痕迹(footprint)，从而提高效率，特别是使用 TBB 分配器时(在多线程应用中)。然而你会很容易失去这个优势，特别是在如下的典型例子中。

1. 你动态的分配类对象，它们的 new/delete 没有使用 Standard::Allocate()/::Free()。
2. 你使用标准容器(std::vector, std::map, std::list,...)。

#问题 1 很容易通过在基类中重新定义 new/delete 操作来解决，正如 OCC 那样做的。

#问题 2 较难一些，因为标准容器，缺省情况下，使用标准的分配器(std::allocator<T>)。这样所有的辅助元素(例如，链表节点)都在 OCC 的管理内存分配器之外分配内存。

这种情况到现在已经可以解决了。这里，我分享一段代码，这段代码使用 ISO C++ 2003 标准实现了标准分配器接口(也符合最近公布的 C++11 标准)。

下载 Standard_StdAllocator.hxx.

这是一个纯头文件，没有 cxx 文件和 lib 文件，只需复制到你的工程中开始使用即可。实现是从 tbb::tbb_allocator 继承来的，它也符合 C++标准的要求。

这个文件可以自由使用，没有版权费用，也可以放在 Public domain，版权具有最大限度的自由。我也希望 OCC 团队乐意将这个文件集成到 OCC 中。

使用的例子如下面的代码所示：

```
typedef Standard_StdAllocator<void> allocator_type;

std::vector<TopoDS_Shape, allocator_type> aSVec;
TopoDS_Solid aSolid = BRepPrimAPI_MakeBox (10., 20., 30.);
aSVec.push_back (aSolid);

std::list<int_Shape, allocator_type> aList;
aList.push_back (1);
```

还有一个单元测试文件(下载 Standard_StdAllocatorTest.cxx, http://www.opencascade.org/org/forum/thread_22234/?forum=3)。代码是基于 Qt 测试框架的，根据自述说明应该可以移植到其他框架。

谁如果有兴趣可以为在 NCollection containers 中使用的 NCollection_BaseAllocator 实现类似的标准分配器，它与这个实现一样直接。

POSTED BY ROMAN LYGIN AT 20:12, 2011-11-24

12 COMMENTS:

1. 

   Thomas Paviot November 25, 2011 12:58 PM

   谢谢 Roman 写了这么有趣的文章，并将自己的工作分享出来。然而好像放文件的地方出问题了。我不能下载 hxx/cxx 文件，但是它们显示出来是 html 代码，与 C++代码没关系。

   Reply

2. 

   Roman Lygin November 25, 2011 4:12 PM

   Hi Thomas,
   请到下面的网址下载：

   http://www.opencascade.org/org/forum/thread_22234/.

   Reply

3. 

   AnonymousNovember 25, 2011 6:11 PM

   Standard_StdAllocator.hxx 的链接好像断了。

   Reply

4. 

Roman Lygin November 26, 2011 10:09 AM
直接点击链接，然后点击下载按钮。

Reply

5. 

Thomas Paviot November 27, 2011 3:46 PM

Hi Roman, 在 OSX/gcc-4.2.1 上出现了编译错误 (参见 https://github.com/tpaviot/oce/issues/204)

Reply

6. 

Roman Lygin November 27, 2011 3:52 PM
Hi Thomas, 谢谢你给我报告这个错误，但是对不起，我只在 VS2008 上面编译过，因为我要带着笔记本到处走。
你能不能检测一下这个错误，然后修正一下。
Roman

Reply

7. 

Thomas Paviot November 27, 2011 10:39 PM
修正已经放到了 OCE(Open CASCADE E...)文件仓库了。

另外，你说“使用一般的内存分配机制可以降低内存的痕迹并且提高性能”。在性能和内存痕迹方面，我们从你的实现包中能得到什么改进。你是不是取得过显著的改进。

Reply

8. 

Roman Lygin November 27, 2011 11:51 PM

谢谢 Denis Barbier 修正这个错误，并将修改集成进来了。

当 OCC 和标准内存分配机制混合后的负载较大时，性能和内存痕迹方面的性能提高较大。我没有做过这两者合并的情况，但是下面的解释会给你提供一些思路。

假如代码大量的使用这两者，只要在 OCC 机制(使用 OCC 或者 TBB 分配器)之外使用标准的分配内存容器，就会有很多的内存使用痕迹，OCC 或者 TBB 会自己保留很多内存。

(标准内存分配的)性能降低来源于两个方面：a) 内存碎片，在通过标准分配器大量为对象分配/释放内存时；b) 多线程环境。在多线程环境中标准内存分配器使用中等粒度的锁，会导致线程相互等待(where standard allocator uses standard allocator uses a central coarse-grain lock causing threads to wait for each other.)。当使用 TBB 分配器时这两个问题都得到了解决，当使用 OCC 时问题 a)得到了解决，但是问题 b)仍然存在。

Reply

9. 

KapilApril 3, 2012 8:42 PM

Hi Roman,

我已经被这个问题困扰了两个星期了——我想要生成一个简单的圆柱面。

然后我采用的方法是生成一个实体圆柱，然后从其中提取出来圆柱面。

然后我用了一些简单的测试，发现环的顺序不对——这怎么可能——Open CASCADE 生成的东西是非法的，我没法进展下去了！！！

你能不能帮我解决这个问题——我提前感谢你花时间解决这个问题。我真的非常需要这个功能，我搜索论坛上的文章，已经有人提出了这是 Open CASCADE 的一个 bug，但是却没有任何帮助。

```
// code

double radius = 2;
double height = 1600;
gp_Pnt P(0, 0, 0);
gp_Vec V(0, 0, 1);
```

```
V.Normalize();
TopoDS_Solid solidCylinder = kernel->CreateSolidCylinder(P, V,
radius, height);

// explore the solid and gets its faces
//kernel->AnalyzeFace(cylface, 0, true);
TopoDS_Face cylindricalFace;
for (TopExp_Explorer it(solidCylinder, TopAbs_FACE);
it.More(); it.Next())
{
const TopoDS_Face &aFace = TopoDS::Face(it.Current());
const Handle(Geom_Surface) &aSurface = BRep_Tool::Surface(aFace);
if (aSurface->DynamicType() ==
STANDARD_TYPE(Geom_CylindricalSurface))
{
cylindricalFace = aFace;
break;
}
}

for (TopExp_Explorer it(cylindricalFace, TopAbs_WIRE);
it.More(); it.Next())
{
const TopoDS_Wire& aWire = TopoDS::Wire(it.Current());
double precision = 1e-04;
ShapeAnalysis_Wire saw(aWire,cylindricalFace, precision);
if(saw.CheckOrder())
throw exception("Not expecting this");
}
//code ends
```

Reply

10. 

Roman Lygin April 3, 2012 8:59 PM

Hi Kapil,

为什么你不直接生成呢：

```
TopoDS_Face aFace = BRepBuilderAPI_MakeFace (new
Geom_CylindricalSurface (gp_Ax3 (P, V, Vx), radius), 0, 2 * PI, 0,
height); //Vx is perpendicular to V and defines X axis
```

希望这对你有帮助！
Roman

Reply

11.

KapilApril 4, 2012 12:46 AM

非常感谢你的回复。

我的帖子可能有点长，请耐心读完。

首先介绍一下我正在做的东西。
我想生成一个圆柱面*face*，然后三角化它。
然后用其他曲面*surfaces* (planar, cylindrical, conical, ...)剪裁。

(译者注：这个问题对我们的学习帮助不大，其中可能包含提问者的错误观点，所以对该问题的翻译省略了。)

或许我上面讲的方法走了远路，但是我仍然想知道为什么得到的顺序出错了。我非常感谢你能给我指条明路。

代码段 1:

```
test_face_trimming_with_bounding_box(TopoDS_Face aShape)
{
gp_Pnt min(-50, -50, -50);
gp_Pnt max(50, 50, 50);

gp_Pnt minO(-5000, -5000, -5000);
gp_Pnt maxO(5000, 5000, 5000);

BRepPrimAPI_MakeBox inner(min, max);
inner.Build();
TopoDS_Solid aInnerBox = inner.Solid();

BRepPrimAPI_MakeBox outer(minO, maxO);
outer.Build();
TopoDS_Solid aOuterBox = outer.Solid();
```

```
BRepAlgoAPI_Cut aSub(aOuterBox, aInnerBox);
aSub.Build();
const TopoDS_Shape& aOutBoxWithHole = aSub.Shape();

TopoDS_Face splitFace;
BRepAlgoAPI_Cut aCut(aShape, aOutBoxWithHole);
aCut.Build();
for (TopExp_Explorer iter(aCut.Shape(), TopAbs_FACE); iter.More();
iter.Next())
{
//expecting a single face
splitFace = TopoDS::Face(iter.Current());
}
}
```

代码段 2：

```
test_bounding_box()
{
gp_Pnt min(-50, -50, -50);
gp_Pnt max(50, 50, 50);

BRepPrimAPI_MakeBox aBox(minPt, maxPt);
aBox.Build();
const TopoDS_Solid& aSolid = aBox.Solid();

for (TopExp_Explorer it(aSolid, TopAbs_WIRE);
it.More(); it.Next())
{
double precision = 1e-04;
ShapeAnalysis_Wire saw(aWire, *aParentFace, precision);
if (saw.CheckOrder())
{
throw exception("Wire not ordered");
}
}
```

再次感谢。

Reply

12.

Shahana Shafiuddin October 6, 2012 9:17 PM

嗯，我不知道这个。

Reply

附录：

1) Standard_StdAllocator.hxx

```
// Author: Roman Lygin, 2011.
// This file is put into Public Domain and thus can freely be used for any purpose.
// The author disclaims any rights and liabilities.

#ifndef _Standard_StdAllocator_HeaderFile
#define _Standard_StdAllocator_HeaderFile

#include <Standard.hxx>

//! Implements allocator requirements as defined in ISO C++ Standard 2003, section 20.1.5.
/*! The allocator uses Open CASCADE Technology memory management API (
      Standard::Allocate() and Standard::Free()). It can be used with standard
      containers (std::vector, std::map, etc) to use centralized Open CASCADE
      memory management and hence decrease memory footprint and/or increase
      performance.

      Example of use:
      \code
      std::vector<TopoDS_Shape, Standard_StdAllocator<TopoDS_Shape> > aVec;
      TopoDS_Solid aSolid = BRepPrimAPI_MakeBox (10., 20., 30.);
      aVec.push_back (aSolid);
      \endcode
*/
template<typename T>
class Standard_StdAllocator {
public:
      typedef typename T value_type;
      typedef value_type* pointer;
      typedef const value_type* const_pointer;
      typedef value_type& reference;
      typedef const value_type& const_reference;
      typedef size_t size_type;
      typedef ptrdiff_t difference_type;
      template<typename U> struct rebind {
            typedef Standard_StdAllocator<U> other;
      };

      //! Constructor.
```

```
/*! Creates an empty object.*/
Standard_StdAllocator() throw() {}

//! Constructor.
/*! Creates an empty object.*/
Standard_StdAllocator( const Standard_StdAllocator& ) throw() {}

//! Destructor.
/*! Empty implementation.*/
    ~Standard_StdAllocator() throw() {}

    //! Constructor.
    /*! Creates an empty object.*/
    template<typename U> Standard_StdAllocator( const Standard_StdAllocator<U>& ) throw()
{}

    //! Returns an object address.
    /*! Returns &x.*/
    pointer address( reference x ) const { return &x; }

    //! Returns an object address.
    /*! Returns &x.*/
    const_pointer address( const_reference x ) const { return &x; }

    //! Allocates memory for \a n objects.
    /*! Uses Standard::Allocate().*/
    pointer allocate( size_type n, const void* /*hint*/ = 0 )
    { return pointer( Standard::Allocate( n * sizeof( value_type ))); }

    //! Frees previously allocated memory.
    /*! Uses Standard::Free().*/
    void deallocate( pointer p, size_type )
    {
        Standard_Address a = p;
        Standard::Free (a); //Standard::Free() requires Standard_Address&
    }

    //! Returns the largest value for which method allocate might succeed.
    size_type max_size() const throw()
    {
        size_type aMax = static_cast<size_type>( -1 ) / sizeof( value_type );
        return (aMax > 0 ? aMax : 1);
    }
```

```
        //! Constructs an object.
        /*! Uses the placement new operator and copy constructor to construct an object.*/
        void construct( pointer p, const_reference val )
        { new( static_cast<void*>( p )) value_type( val ); }

        //! Destroys the object.
        /*! Uses object destructor.*/
        void destroy( pointer p ) { p->~value_type(); }
};

//! Implements specialization Standard_StdAllocator<void>.
/*! Specialization is similar to std::allocator<void>, as defined in ISO C++
    Standard 2003, section 20.4.1.

    This specialization is interchangeable with other ones.

    Example of use:
    \code
    std::vector<TopoDS_Shape, Standard_StdAllocator<void> > aVec;
    TopoDS_Solid aSolid = BRepPrimAPI_MakeBox (10., 20., 30.);
    aVec.push_back (aSolid);
    \endcode
*/
template<>
class Standard_StdAllocator<void> {
public:
    typedef void* pointer;
    typedef const void* const_pointer;
    typedef void value_type;
    template<typename U> struct rebind {
        typedef Standard_StdAllocator<U> other;
    };
};

template<typename T, typename U>
inline bool operator==( const Standard_StdAllocator<T>&, const Standard_StdAllocator<U>& )
{ return true; }

template<typename T, typename U>
inline bool operator!=( const Standard_StdAllocator<T>&, const Standard_StdAllocator<U>& )
{ return false; }


#endif
```

2) Standard_StdAllocatorTest.cxx

```
#include <Standard_StdAllocator.hxx>

class Standard_StdAllocatorTest: public QObject
{
	Q_OBJECT

private slots:

	void Traits();
	void Containers();
};



void Standard_StdAllocatorTest::Traits()
{
	//type definitions
	typedef Handle_Standard_Transient elem_type;
	typedef Standard_StdAllocator<elem_type> allocator_type;

	QCOMPARE (sizeof (allocator_type::value_type),sizeof (elem_type));
	QCOMPARE (sizeof (allocator_type::pointer),sizeof (void*));
	QCOMPARE (sizeof (allocator_type::const_pointer),sizeof (void*));

	elem_type aDummy;
	allocator_type::reference aRef = aDummy;
	allocator_type::const_reference aConstRef = aDummy;

	QCOMPARE (sizeof (allocator_type::size_type),sizeof (size_t));
	QCOMPARE (sizeof (allocator_type::difference_type),sizeof (ptrdiff_t));

	typedef int other_elem_type;
	QCOMPARE  (sizeof  (allocator_type::rebind<other_elem_type>::other::value_type),  sizeof
(other_elem_type));
}

void Standard_StdAllocatorTest::Containers()
{
	//typed allocator
	std::list<TopoDS_Shape, Standard_StdAllocator<TopoDS_Shape> > aL;
	TopoDS_Solid aSolid = BRepPrimAPI_MakeBox (10., 20., 30.);
```

```
  aL.push_back (aSolid);
  QCOMPARE (aL.size(), size_t (1));

  //type cast
  Standard_StdAllocator<char> aCAlloc;
  std::vector<int, Standard_StdAllocator<int> > aV (aCAlloc);
  aV.push_back (1);
  QCOMPARE (aV.size(), size_t (1));

  //using void-specialization allocator
  std::vector<int, Standard_StdAllocator<void> > aV2;
  aV2.resize (10);
  aV2.push_back (-1);
  QCOMPARE (aV2.size(), size_t (11));
}
```