



OpenEye
Scientific Software

Shape TK – C++
Release 1.8.1

OpenEye Scientific Software, Inc.

January 11, 2012

CONTENTS

1	Front Matter	1
2	OEShape Theory	3
2.1	Theory	3
3	OEShape Examples	7
3.1	Calculating simple overlap	7
3.2	Color Overlap	9
3.3	Best Overlay	14
3.4	Calculating NxN scores	22
3.5	Calculating shape characteristics	24
4	API	27
4.1	OEShape Classes	27
4.2	OEShape Constants	48
4.3	OEShape Functions	51
5	Release Notes	59
5.1	ShapeTK 1.8.1	59
5.2	ShapeTK 1.8.0	59
5.3	ShapeTK 1.7.2	60
5.4	ShapeTK 1.7.1	60
5.5	ShapeTK 1.7.0	61
5.6	ShapeTK 1.6.2	62
5.7	ShapeTK 1.6.1	62
5.8	ShapeTK 1.6	62
5.9	ShapeTK 1.5	63
5.10	Indices and tables	63
	Bibliography	65
	Index	67

FRONT MATTER

Copyright 1997-2012 OpenEye Scientific Software, Santa Fe, New Mexico. All rights reserved.

All rights reserved. This material contains proprietary information of OpenEye Scientific Software. Use of copyright notice is precautionary only and does not imply publication or disclosure.

The information supplied in this document is believed to be true but no liability is assumed for its use or the infringement of the rights of others resulting from its use. Information in this document is subject to change without notice and does not represent a commitment on the part of OpenEye Scientific Software.

This package is sold/licensed/distributed subject to the condition that it shall not, by way of trade or otherwise, be lent, re-sold, hired out or otherwise circulated without OpenEye Scientific Software's prior consent, in any form of packaging or cover other than that in which it was produced. No part of this manual or accompanying documentation, may be reproduced, stored in a retrieval system on optical or magnetic disk, tape, CD, DVD or other medium, or transmitted in any form or by any means, electronic, mechanical, photocopying recording or otherwise for any purpose other than for the purchaser's personal use without a legal agreement or other written permission granted by OpenEye.

This product should not be used in the planning, construction, maintenance, operation or use of any nuclear facility nor the flight, navigation or communication of aircraft or ground support equipment. OpenEye Scientific Software, shall not be liable, in whole or in part, for any claims arising from such use, including death, bankruptcy or outbreak of war.

Windows is a registered trademark of Microsoft Corporation. Apple, OS X, and Macintosh are registered trademarks of Apple Computer, Inc. AIX and IBM are registered trademarks of International Business Machines Corporation. UNIX is a registered trademark of the Open Group. RedHat is a registered trademark of RedHat, Inc. Linux is a registered trademark of Linus Torvalds. SPARC is a registered trademark of SPARC International Inc.

SYBYL is a registered trademark of TRIPOS, Inc. MDL is a registered trademark and ISIS is a trademark of Accelrys, Inc. SMILES, SMARTS, and SMIRKS may be trademarks of Daylight Chemical Information Systems. Macromodel is a trademark of Schrodinger, Inc.

Python is a trademark of the Python Software Foundation. Java is a trademark or registered trademark of Sun Microsystems, Inc. in the U.S. and other countries.

Other products and software packages referenced in this document are trademarks and registered trademarks of their respective vendors or manufacturers.

OESHAPE THEORY

2.1 Theory

2.1.1 Molecular Shape

What do we mean by shape? The word is often used without consideration of precise meaning but in this document we shall be very clear as to the definition of shape. Two entities will have the same shape if their volumes exactly correspond. The more the volumes differ, the more the shapes will differ. We will give a precise mathematical exposition below, but it is worth noting even at this most basic level shape is defined as a relative quantity, depending on references to other shapes. In this we differ from approaches that attempt to provide absolute, canonical, *shapes* by which to categorize molecules.

What do we mean by *volume*? A volume is any scalar field. This means a function that has a single number, or *scalar*, value at each point in space. The *special case* for the common understanding of volume is a specific scalar field that has a value of one inside an object and zero outside. The volume of a scalar field is:

$$V(\text{volume}) = \int f(x, y, z)dv$$

The volume function, **f**, is also referred to as the *characteristic* function. When the characteristic function corresponds to the common definition of a volume field this integral corresponds to what is commonly expected by volume. However, we are not restricted to such simple functions and can still calculate a **V**. In general the volume of a scalar field is a *contraction* of the information represented by that characteristic function. It is more precisely referred to as the zeroth-order contraction, or *moment*. We will discuss other moments and their uses later, but one immediate observation is that two objects can not have the same shape if their volumes are not the same. The converse is obviously not true. Rather, two objects can have the same volume and not have the same shape. Volume is typical, therefore, of most contractions of information.

We can now write down a precise definition of shape similarity. Consider the integral:

$$S_1 = \int |f(x, y, z) - g(x, y, z)|dV$$

where **f** and **g** are different characteristic functions. If this integral is zero then **f** and **g** are actually the same function and therefore correspond to the same shape. The larger the integral, the more different the shapes defined by **f** and **g**. It defines a metric quantity between the two fields **f** and **g**. The word *metric* is used loosely to mean *shape*, but here we mean the precise mathematical definition: *i.e.* a distance that is 1) always positive, 2) zero if and only if two entities are identical and 3) that obeys the triangle inequality. The triangle inequality states that if entity A is distance *x* from entity B and B is distance *y* from entity C then the distance between A and C is bounded by $|x-y|$ and $|x+y|$. The type of comparison shown in S_1 is referred to as an L_1 metric. Another metric is the S_2 metric:

$$S_2 = \sqrt{\int [f(x, y, z) - g(x, y, z)]^2 dV}$$

Multiplying the terms in the integral out gives:

$$S_2^2 = \int f(x, y, z)^2 dV + \int g(x, y, z)^2 dV - 2 \int f(x, y, z)g(x, y, z) dV$$

This is the fundamental equation for shape comparison. We rewrite it as:

$$S_{f,g} = I_f + I_g - 2O_{f,g}$$

The I terms are the self-volume overlaps of each entity (for our purposes - molecule), while the O term is the overlap between the two functions. They constitute the three terms we need to compare the shapes of two fields. The I terms are independent of orientation but not O . Finding the orientation that maximizes O , and hence minimizes $S_{\{f,g\}}$, is equivalent to finding the best overlay between the two objects (a quantity that has its own, distinct metric properties). We also note here that the quantity referred to as a Tanimoto coefficient may be derived by recombining I 's and O so:

$$Tanimoto_{f,g} = \frac{O_{f,g}}{I_f + I_g - O_{f,g}}$$

Tanimoto coefficients will be familiar to those who use them for bitvector fingerprint comparison. An alternative measure is the Tversky coefficient, also mostly used for similarity between bitvector fingerprints. Similarly to the Tanimoto coefficient above, we can define a shape Tversky measure. The base equation for the Tversky coefficient is:

$$Tversky_{f,g} = \frac{O_{f,g}}{\alpha I_f + \beta I_g}$$

Normally, $\alpha + \beta = 1$, and for our current use, α is chosen to be 0.95. Since this introduces an asymmetry, the Tversky calculation depends on which molecule's self-overlap has the α pre-factor. ROCS calculates two Tversky values, one with the query molecule with α as the pre-factor and a second with the database molecule with α as the pre-factor. Also, note that since shape is a field property, instead of a simple scalar like a bitvector, shape Tversky can be larger than 1.0 since the overlap $O_{\{f,g\}}$ can be larger than a molecule's self-overlap, I_f .

The OpenEye Shape Toolkit is a set of calculational objects designed to facilitate the calculation of these field-metric quantities. ROCS is an application built on top of the Shape toolkit.

2.1.2 Shape Characteristics and the Use of Gaussians

Molecules are traditionally viewed as a set of fused spheres, sometimes referred to as the CPK model. The common view of molecular volume is then of a characteristic function that is one (1) inside at least one sphere and zero (0) outside. How do we calculate the volume of such a seemingly simple function? The volume of a single sphere is $(4\pi r^3)/3$ but the complication for two fused spheres is that we have to account for the shared volume and not count it twice. For more than two atoms, there are triple intersections that must be added back in if we have removed the three pairs of intersections. The general formula for N spheres that explicitly calculates the volume of every level of overlap and its correct contribution is:

$$V = 1 - \int \prod_i^N (1 - f_i) dv$$

This is easy to write, not so easy to solve because the analytic formulae for overlaps of increasing order are highly non-trivial (although they have been derived to arbitrary order). It is fair to say that this has hindered the development of shape comparison in many ways. Attempts to use analytic formulae led to very slow programs and approximate methods, for instance using grids of points that are turned *in* or *out* by each sphere, do not give smooth gradients required for minimization. Brian Masek (AstraZeneca) was the first to attempt to optimize overlaps of molecules using the analytic approach. His program would take several minutes per minimization. In addition it would often suffer from a common problem when using functions that vary sharply (such as solid spheres): it would often get stuck in local minima. Nevertheless, Brian did have encouraging success using this method to find similarities not obvious from chemical structure.

The conceptual breakthrough in shape comparison came in 1995 in a paper by Andrew Grant (AstraZeneca) and Barry Pickup (University of Sheffield) ([Grant-1995], [Grant-1996], [Grant-1997]). They showed that if one let go of the concept of the characteristic function being binary, and instead use a sum of continuous functions, *i.e.* a Gaussian, that the solid-sphere volume, could be recovered to high accuracy (typically ~0.1%). A sphere has one defining parameter, its radius, whereas a Gaussian has two defining parameters, its prefactor, p , and its width, w :

$$pe^{-wx^2}$$

Grant and Pickup found that by fixing p to 2.7 and setting w for each atom such that the volume integral for each atom agreed with its solid-sphere volume, they achieved remarkable precision. In addition, the overlap terms between any two atoms, and hence any higher-order overlaps, are all Gaussian functions themselves because of the Gaussian Contraction formula (shown here for one spatial variable):

$$\int e^{a(x-x_i)^2} e^{b(x-x_i)^2} = \int e^{(a+b)(x-x_i)^2}$$

i.e. two atomic-Gaussians overlap to produce another Gaussian. Likewise, a three atomic-Gaussian overlap is that of an overlap-Gaussian with an atomic-Gaussian, hence another Gaussian. The simplicity of these formulae and the formula for the volume of each individual Gaussian leads to very efficient algorithms for the calculation of the volume of a molecule so represented (the OpenEye method calculates several thousand volumes per second while calculating intersections up to sixth order).

In addition to simple calculation of molecular volume, which is the zeroth-order moment of the characteristic function, the ease of evaluation of intersections allows for accurate calculation of high-order moments: called the *steric multipoles*. For instance, if the product formulae for atomic and intersection Gaussians yields n Gaussians, the first order moments are:

$$M_{1,x} = \sum_{i=1}^n \int x e^{a_i|(x-x_i)^2+(y-y_i)^2+(z-z_i)^2|}$$

$$M_{1,y} = \sum_{i=1}^n \int y e^{a_i|(x-x_i)^2+(y-y_i)^2+(z-z_i)^2|}$$

$$M_{1,z} = \sum_{i=1}^n \int z e^{a_i|(x-x_i)^2+(y-y_i)^2+(z-z_i)^2|}$$

These integrals are easy to solve and their sum can be set to zero by an appropriate choice of origin: the center of *mass* for the sum of Gaussians. Second-order moments are found from integrals of the type:

$$M_{2,PQ} = \sum_{i=1}^n \int PQ e^{a_i|(x-x_i)^2+(y-y_i)^2+(z-z_i)^2|}$$

where P and Q are chosen from (x,y,z), *e.g.* x^2 , xy etc.

These moments can be thought of as a symmetric 3*3 matrix which we refer to as the *mass matrix*. Rotating or translating the molecule will change the moments and the transform that sets the first-order moments to zero and diagonalizes the mass-matrix puts the molecule into its *inertial frame*. By convention we assign the x-axis to the largest eigenvector of the mass matrix, y-axis to the median eigenvector, z-axis to the smallest. Note that this orientation is still not uniquely defined: 180 degree rotations around any axis also diagonalize the mass-matrix. The eight (2^3) possible transforms that can be generated by combinations of such rotations actually lead to four unique inertial orientations.

If a molecule is aligned to its inertial frame, all higher-order steric multipoles become invariant, ignoring certain sign-changes from the four-fold degeneracy of the inertial frame. As such they, as well as the second-order moments, are shape *descriptors*. They are still contractions of the information contained in the characteristic field, *i.e.* two molecules can have the same steric moments and yet have different shapes. (Moments are *complete* in that if we calculate them to infinite order they do exactly define volume but this is seldom a practical approach!) Nevertheless, they do contain useful information and can be used as a rapid, approximate, filter for shape similarity.

The same advantages that allow for the calculation of molecular volume carry over to the calculation of molecular volume overlap. The overlap of volumes are Gaussian contractions, easily tabulated and efficiently retrieved. Andy rewrote Brian's program and obtained an order-of-magnitude improvement in performance as well as another remarkable observation: if the starting orientation of each molecule is that given from the inertial frame then very few "false" minima are produced. The smoothness of the Gaussian characteristic function is enough to overcome the problems with convergence in Brian's program. The four possible "inertial" starting points were enough to find the best, global, overlay between two molecules. This observation and the Gaussian approach are the basis of the OpenEye Shape Toolkit and ROCS program for rapid shape overlay.

But note, despite the algorithmic advantages, a correlation with common perception has been lost. Because the pre-factor of each atomic Gaussian is not unity, the characteristic function does not correspond to the inside/outside description with which we are most comfortable. In the Gaussian model all points in space are to some degree inside and to some degree outside. That is, the Gaussian model typically shows about 0.1% error with respect to the solid sphere model due to the fact that it includes a portion of all points in space inside the volume.

OESHAPE EXAMPLES

3.1 Calculating simple overlap

The simplest object in the Shape toolkit, `OEOverlap`, is used to calculate simple, static, overlap between two objects (molecules or grids). Note that static means that the two input species (ref and fit) are not moved at all. This object simply calculate the overlap given the input positions. Performing calculations that actually optimize the alignment/overlap are done with the `OEBestOverlay` object (see *Theory* chapter).

This first example reads in a reference molecule and a few fit molecules and prints out the overlap calculated. Note that this example uses all default settings for `OEOverlap` (discussed in the following sections).

Listing 1: Simple overlap using Exact Overlap

```
/*  
  Copyright (C) 2005,2006,2007,2008 OpenEye Scientific Software, Inc.  
*/  
#include "openeye.h"  
  
#include "oeplatform.h"  
#include "oesystem.h"  
#include "oechem.h"  
#include "oeshape.h"  
  
using namespace OEPlatform;  
using namespace OESystem;  
using namespace OEChem;  
using namespace OEShape;  
using namespace std;  
  
int main(int, char**)  
{  
    oemolistream reffs("a.mol2");  
    oemolistream fitfs("rocs_hits_1.sdf");  
  
    OEGraphMol refmol;  
    OEReadMolecule(reffs, refmol);  
  
    OEOverlap ov;  
    ov.SetRefMol(refmol);  
}
```

```
OEOverlapResults res;
OEGraphMol fitmol;
while (OEReadMolecule(fitfs, fitmol))
{
    oeout << fitmol.GetTitle();
    ov.Overlap(fitmol, res);
    oeout << " exact tanimoto = " << res.tanimoto << oeendl;
}
return 0;
}
```

3.1.1 Overlap Methods

This nature of the algorithm used to calculate overlap is determined by the `OEOverlap::SetMethod` member of `OEOverlap`. At present there are four types of algorithms implemented: *Exact*, *Analytic*, *Analytic2*, and *Grid*. These are defined in the `OEOverlapMethod` namespace.

The *Exact* option is as detailed above: all pairs are considered and the Gaussian overlaps are calculated exactly. The two *Analytic* options use an approximation to the overlap between two Gaussians that is accurate to about one part in a thousand. In addition, *Analytic2* uses a proximity grid to only calculate those atoms pairs that are within a certain threshold distance (by default 4.5 Å). This approximation adds another one part in a thousand average error but is faster for larger molecules. The final option, *Grid*, uses a grid representation of the volume of the target molecule. It requires significant set-up time relative to the cost of a single overlap calculation (~0.01s compared to ~0.0001s) but is significantly faster than other methods for the evaluation of each overlap once set. *Grid* suffers a few caveats and drawbacks. First is that, currently, although reference radii are all treated as given, fit atoms are all treated as if they have one radius (that assigned to carbon, and setable via `OEOverlap::SetCarbonRadius`). The second is that the approximation is slightly worse, typically a few parts in a thousand, at typical grid resolutions. Both *Analytic2* and *Grid* improve performance when the fit molecule is large (>20 atoms) because, if there are **n** atoms in the fit and **m** in the reference, the work per atom in the fit is proportional to a constant not **n**.

By default, `OEOverlap` performs an *Exact* overlap calculation. The `OEBestOverlay` object (used in ROCS) uses *Grid* for speed.

3.1.2 Radii approximations

Since we are considering molecular volume overlap as a measure, the radii used for each atom is important. There are essentially two settings. A radii approximation of `All` means each atom will be treated with the radius as passed in. Alternatively, one can treat all heavy atoms as similar and apply the `Carbon` radius approximation. With this, all heavy atoms will be assigned the same radius, regardless of the value attached to each atom. As noted above, if the *Grid* method is chosen, the `Carbon` radius approximation is also turned on.

3.1.3 Using hydrogens

Shape is a heavy atom property and most OpenEye uses (as well as the defaults for `OEOverlap` and `OEBestOverlay`) ignore hydrogens. Hydrogens can be present or not, if `OEOverlap` is told to not use them, they will not affect the overlap scores.

3.1.4 Adding scores to molecules using SD data

This next example program switches on the *Analytic* overlap method and also uses a little extra `OEChem` to attach the overlap scores to each molecule as SD data. This can be used to rescore a set of ROCS hits or to measure the overlap of ROCS alignments against an exclusion region in the binding site.

Listing 2: Rescoring pre-aligned structures

```

/*****
Copyright (C) 2005-2010 OpenEye Scientific Software, Inc.
*****/
#include "openeye.h"

#include "oeplatform.h"
#include "oesystem.h"
#include "oechem.h"
#include "oeshape.h"

using namespace OEPlatform;
using namespace OESystem;
using namespace OEChem;
using namespace OEShape;
using namespace std;

int main(int argc, char *argv[])
{
    if (argc!=4)
        OEThrow.Usage("rescore <reffile> <rocs_hits_file> <output.sdf>");

    oemolistream reffs(argv[1]);
    oemolistream fitfs(argv[2]);
    oemolistream outfs(argv[3]);

    OEGraphMol refmol;
    OEReadMolecule(reffs, refmol);

    OEOverlap ov;
    ov.SetMethod(OEOverlapMethod::Analytic);
    ov.SetRefMol(refmol);

    OEOverlapResults res;
    OEGraphMol fitmol;
    while (OEReadMolecule(fitfs, fitmol))
    {
        ov.Overlap(fitmol, res);
        char buffer[12];
        sprintf(buffer, "%-6.3f", res.tanimoto);
        OESetSDData(fitmol, "AnalyticShapeTanimoto", buffer);
        OEWwriteMolecule(outfs, fitmol);
    }
    return 0;
}

```

3.2 Color Overlap

3.2.1 Overlap of color atoms

As a way of comparing chemical functionality, the Shape toolkit also provides a color overlap function, `OEColorOverlap`. The underlying algorithm is similar to pure shape, but instead of overlapping all heavy atoms, in this case we overlap similarly tagged *color* atoms. The decision of what becomes a color atom is done using SMARTS

pattern matching. And the radius and weight of color interactions is completely user-definable.

3.2.2 Using built-in color force fields

Two color force fields, `ImplicitMillsDean` and `ExplicitMillsDean`, are built into the Shape toolkit. (The `OIColorForceField` class is defined in the next section.) Both of these force fields define similar 6 TYPE color force-fields. The types are hydrogen-bond donors, hydrogen-bond acceptors, hydrophobes, anions, cations, and rings. The `ImplicitMillsDean` force field is recommended.

`ImplicitMillsDean` includes a simple pKa model that assumes pH=7. It defines cations, anions, donors and acceptors in such a way that they will be assigned the appropriate value independent of the protonation state in the reference or fit molecule. For example, if a molecule contains a carboxylate, `ImplicitMillsDean` will consider it an anionic center independent of whether it is protonated or deprotonated. This is convenient for searching databases which have not had careful curation of their protonation states. The `ExplicitMillsDean` file has a similar overall interaction model, however, it does not include a pKa model. It interprets the protonation and charge state of each molecule exactly. Thus, if a sulfate is protonated and neutral, it will not be considered an anion.

The hydrogen-bond models in both `ImplicitMillsDean` and `ExplicitMillsDean` are extensions of the original model presented by Mills and Dean [MillsDean-1996]. They both have donors and acceptors segregated into strong, moderate and weak categories.

Listing 3: Calculating color score

```
/*  
Copyright (C) 2005,2006,2007,2008 OpenEye Scientific Software, Inc.  
*/  
#include "openeye.h"  
  
#include "oeplatform.h"  
#include "oesystem.h"  
#include "oechem.h"  
#include "oeshape.h"  
  
using namespace OEPlatform;  
using namespace OESystem;  
using namespace OEChem;  
using namespace OEShape;  
using namespace std;  
  
int main(int, char**)  
{  
    oemolistream reffs("a.mol2");  
    oemolistream fitfs("rocs_hits_1.sdf");  
  
    OEGraphMol refmol;  
    OEReadMolecule(reffs, refmol);  
  
    OIColorOverlap ov;  
    ov.SetColorForceField(OIColorFFType::ImplicitMillsDean);  
    ov.SetRefMol(refmol);  
  
    OIColorResults res;  
    OEGraphMol fitmol;  
    while (OEReadMolecule(fitfs, fitmol))  
    {
```

```

    oeout << fitmol.GetTitle();
    ov.ColorScore(fitmol, res);
    oeout << " color score = " << res.colorscore << oeendl;
}
return 0;
}

```

This next example uses the ImplicitMillsDean force field to rescore a set of ROCS hits and add the color scores to SD tags.

Listing 4: Using color to add scores to pre-aligned molecules.

```

/*****
Copyright (C) 2005-2010 OpenEye Scientific Software, Inc.
*****/
#include "openeye.h"

#include "oeplatform.h"
#include "oesystem.h"
#include "oechem.h"
#include "oeshape.h"

using namespace OEPlatform;
using namespace OESystem;
using namespace OEChem;
using namespace OEShape;
using namespace std;

int main(int argc, char *argv[])
{
    if (argc!=3)
        OThrow.Usage("colorscore <reffile> <overlaid_file>");

    oemolistream reffs(argv[1]);
    oemolistream fitfs(argv[2]);
    oemolostream outfs(argv[3]);

    OEGraphMol refmol;
    OEReadMolecule(reffs, refmol);

    OEColorOverlap ov;
    ov.SetColorForceField(OEColorFFType::ImplicitMillsDean);
    ov.SetRefMol(refmol);

    cout << "Ref. Title: " << refmol.GetTitle() << " ";
    cout << "Self color: " << ov.GetSelfColor() << endl;

    OEColorResults res;
    OEGraphMol fitmol;
    while (OEReadMolecule(fitfs, fitmol))
    {
        ov.ColorScore(fitmol, res);
        cout << "Fit Title: " << fitmol.GetTitle() << " ";
        cout << "Color Tanimoto: " << res.GetTanimoto() << endl;
    }
}

```

```
    return 0;
}
```

3.2.3 OEColorForceField

The color force field (CFF) can be used to measure chemical complementarity, and to refine shape-based superpositions based on chemical similarity. The CFF is composed of SMARTS rules that determine chemical centers plus rules to determine how such centers interact. In addition to the built-in color force fields, there are several methods for user-defined force fields.

User-defined interactions

As a step toward writing a complete color force field, it is possible to combine built-in rules for color atom assignment with user defined interactions. A new `OEColorForceField` object can be created using one of the built-in types, then the interactions can be cleared using `OEColorForceField::ClearInteractions` and subsequent user interactions added with `OEColorForceField::AddInteraction`.

For example, to use the `ImplicitMillsDean` atom typing rules, but to only consider donor-donor and acceptor-acceptor interactions, one can use the following:

Listing 5: Using `ImplicitMillsDean` with user interactions.

```
/*
*****
Copyright (C) 2005,2006,2007,2008 OpenEye Scientific Software, Inc.
*****
*/
#include "openeye.h"

#include "oeplatform.h"
#include "oesystem.h"
#include "oechem.h"
#include "oeshape.h"

using namespace OEPlatform;
using namespace OESystem;
using namespace OEChem;
using namespace OEShape;
using namespace std;

int main(int argc, char *argv[])
{
    if (argc!=4)
        OEThrow.Usage("usercolor <reffile> <rocs_hits_file> <output.sdf>");

    oemolistream reffs(argv[1]);
    oemolistream fitfs(argv[2]);
    oemolostream outfs(argv[3]);

    OEGraphMol refmol;
    OEReadMolecule(reffs, refmol);

    OEColorForceField cff;
    cff.Init(OEColorFFType::ImplicitMillsDean);
    cff.ClearInteractions();
}
```

```

unsigned int donorType = cff.GetType("donor");
unsigned int acceptType = cff.GetType("acceptor");
cff.AddInteraction(donorType, donorType, "gaussian", -1.0, 1.0);
cff.AddInteraction(acceptType, acceptType, "gaussian", -1.0, 1.0);

OEColorOverlap ov;
ov.SetColorForceField(cff);
ov.SetRefMol(refmol);

oeerr << "Ref. Title: " << refmol.GetTitle() << " ";
oeerr << "Self color: " << ov.GetSelfColor() << oeendl;

OEColorResults res;
OEGraphMol fitmol;
while (OEReadMolecule(fitfs, fitmol))
{
    ov.ColorScore(fitmol, res);
    char cbuffer[10], sbuffer[10];
    sprintf(cbuffer, "%-6.3f", res.colorscore);
    OESetSDDData(fitmol, "MyColorScore", cbuffer);
    sprintf(sbuffer, "%-6.3f", res.scaledcolor);
    OESetSDDData(fitmol, "MyScaledColor", sbuffer);
    OEWriteMolecule(outfs, fitmol);

    oeerr << "Fit Title: " << fitmol.GetTitle() << " ";
    oeerr << "Color Score: " << cbuffer << " ";
    oeerr << "Scaled Color: " << sbuffer << oeendl;
}
return 0;
}

```

Writing color force field files (CFF)

As an alternative to the built-in force fields, the user can define a new color force field using the format described in this section.

The following is a simplified example of a color force field specification.

```

DEFINE hetero [#7,#8,#15,#16]
DEFINE notNearHetero [!#1;!$($hetero);!$(*[$hetero])]
#
#
TYPE donor
TYPE acceptor
TYPE rings
TYPE positive
TYPE negative
TYPE structural
#
#
PATTERN donor [$hetero;H]
PATTERN acceptor [#8&!$(*~N~[OD1]),#7&H0;!$([D4]);!$([D3]-*~, :[$hetero])]
PATTERN rings [R]~1~[R]~[R]~[R]1
PATTERN rings [R]~1~[R]~[R]~[R]~[R]1
PATTERN rings [R]~1~[R]~[R]~[R]~[R]~[R]1
PATTERN rings [R]~1~[R]~[R]~[R]~[R]~[R]~[R]1

```

```
PATTERN positive [+,$([N;!$(**=O)])]
PATTERN negative [-]
PATTERN negative [OD1+0]-[!#7D3]~[OD1+0]
PATTERN negative [OD1+0]-[!#7D4](~[OD1+0])~[OD1+0]
PATTERN structural [$notNearHetero]
#
#
INTERACTION donor donor attractive gaussian weight=1.0 radius=1.0
INTERACTION acceptor acceptor attractive gaussian weight=1.0 radius=1.0
INTERACTION rings rings attractive gaussian weight=1.0 radius=1.0
INTERACTION positive positive attractive gaussian weight=1.0 radius=1.0
INTERACTION negative negative attractive gaussian weight=1.0 radius=1.0
INTERACTION structural structural attractive gaussian weight=1.0 radius=1.0
```

There are four basic keywords in a cff file: **DEFINE**, **TYPE**, **PATTERN**, and **INTERACTION**. The **TYPE** field can be any user-defined term. **TYPES** can be any user-specified string such as “donor”, “acceptor”, “lipophilic anion” etc. The **PATTERN** keyword is used to associate SMARTS patterns with these types. There is no restriction on the number of patterns that can be associated with a user defined type. The position in Cartesian space of the **PATTERN** is taken as the average of the coordinates of the atoms that match the SMARTS pattern. If the desired location of the **PATTERN** is on a single atom of a larger SMARTS pattern recursive SMARTS (written as [$\$(SMARTS)$]) can be used to this effect. Only the first atom in a recursive SMARTS pattern ‘matches’ the molecule, and the rest of the SMARTS pattern defines an environment. By writing a SMARTS pattern in recursive notation the location of the **PATTERN** will be taken as the atomic position of the first matching atom in the pattern. In order to simplify both reading and writing SMARTS, intermediate SMARTS can be associated with words using the **DEFINE** keyword. Once defined, these words can then be used as atom primitives in subsequent SMARTS patterns with the $\$$ prefix (see “DEFINE hetero” and “PATTERN donor” above).

Interactions between types are associated with the **INTERACTION** keyword. Two user-defined **TYPES** must be listed, and whether their interaction is attractive or repulsive. By default the interaction decays as a Gaussian of weight 1.0 Å and radius (decay to 1/e) 1.0 Å. The weight and radius can be modified by keywords **WEIGHT** and **RADIUS**. At present, the only alternative to a Gaussian decay is invoked by the **DISCRETE** keyword. A discrete interaction contributes all of **WEIGHT** if the inter-type distance is less than **RADIUS**, or zero. Since it is not differentiable it makes no contribution to optimization (*i.e.* because the gradient of a **DISCRETE** function is 0 or infinite).

3.3 Best Overlay

OEBestOverlay is the object used to maximize shape (and color) overlap between two objects. OEBestOverlay uses the same overlap function and color function as described earlier, but it optimizes the overlap between the reference and fit object. For speed, the default overlap settings for OEBestOverlay are to use the `OEOverlapMethod::Grid` overlap method (which implies the `OEOverlapRadii::Carbon` radius approximation) and hydrogens are ignored.

3.3.1 Starting positions for optimization

Since OEBestOverlay performs an optimization, the starting point of the optimization is important. By default, OEBestOverlay uses an inertial frame alignment method to decide on starting positions (`OEBOrientation::Inertial`). The reference structure is aligned by its principal moments of inertia, then the fit object is aligned in 4 positions with the primary and secondary moments of inertia in both possible directions.

Thus, for any given optimization, there are 4 results returned, usually only one of which is useful. In order to deal with structures with symmetrical moments of inertia, OEBestOverlay may perform additional starting points. For a reference or fit where the 2 major moments of inertia are equal (to a user-defined percent, nominally 15%), 4 extra

starting positions are generated. In the rare case of a molecule with all 3 moments of inertia essentially equal, 20 random starting translations and rotations are chosen as starting positions.

For virtual screening uses, where the reference and fit molecule are similar in size, `OEBOOrientation::Inertial` provides an excellent choice for starting positions that balancing quality results with speed. But, there are times when there is a large difference in size and a more elaborate search is needed. For these, there are a couple of built-in searches as well a user-defined search. To increase searching, instead of doing the inertial frame rotations with to 2 molecule centers-of-mass (COM) aligned, we can do a set of inertial rotations at many more locations. Using `OEBOOrientation::InertialAtHeavyAtoms`, `OEBestOverlay` will translate the COM of the fit molecule to each heavy atom of the reference molecule. At each of these translations, it will perform the 4 basic inertial transforms. Note that this means that instead of 4 (or 8) starting positions, there will be 4 x number of reference molecule heavy atoms starts, resulting in 20-30 more starting positions compared to the default. Obviously this will slow the overall calculation so this is not recommended for high-volume virtual screening, but is very useful for low-volume fragment searching.

A slightly less aggressive set of starting positions (`OEBOOrientation::InertialAtColorAtoms`) translates the COM of the fit molecule to each color atom position and does the 4 inertial rotations. For an average reference molecule with ~10 color atoms, this will be faster than using all heavy atom positions, but not as elaborate of a search.

Finally, they user can set `OEBOOrientation::UserInertialStarts` and then provide a set of coordinates to `OEBestOverlay::SetUserStarts`. `OEBestOverlay` will then use each of these absolute coordinates as a place to translate the COM of the fit molecule and do the inertial transforms.

Alternatively, the user may desire to start the optimization from a single, pre-aligned position. For this case, the user can specify `OEBOOrientation::AsIs` as a starting position.

And finally, there is a method to generate **N** random starting positions where **N** is user-defined as well as the maximum translation allowed between random starts.

In most cases, the default `OEBOOrientation::Inertial` frame starting positions are completely sufficient to find optimal overlap.

Note that the fit object is not moved during the optimization. Part of the results returned from the calculation are the transforms necessary to move the fit object into the final orientation. This allows the user to skip this step if only scores are desired. It also allows application of the same transform to other objects.

3.3.2 Ways to deal with `OEBestOverlay` results

Unlike `OEOverlap` and `OEColorOverlap`, `OEBestOverlay` uses multi-conformer molecules as the reference and fit (mostly for efficiency). As such, a single `OEBestOverlay` calculation can return numerous results. Basically, each calculation will return **N** results where **N** is the number of ref conformers times the number of fit conformers times the number of starting positions for each pair. So comparing 2 molecules with 10 conformers each could return 400 or more results.

There are two helper classes designed solely to contain these results and to make it easy to extract all or just the desired subset. `OEBestOverlayResults` holds the results of a single pair of conformers. It contains a set of `OEBestOverlayScore` objects, one for each starting position.

The first example, uses 2 iterators to show all the results.

Listing 6: Getting all the scores from `OEBestOverlay`.

```

/*****
Copyright (C) 2005-2010 OpenEye Scientific Software, Inc.
*****/
#include "openeye.h"

```

```
#include "oeplatform.h"
#include "oesystem.h"
#include "oechem.h"
#include "oeshape.h"

using namespace OEPlatform;
using namespace OESystem;
using namespace OEChem;
using namespace OEShape;
using namespace std;

int main(int argc, char *argv[])
{
    if (argc!=3)
        OThrow.Usage("bestoverlay1 <reffile> <fitfile>");

    oemolistream reffs(argv[1]);
    oemolistream fitfs(argv[2]);

    OEMol refmol;
    OEReadMolecule(reffs, refmol);

    OEBestOverlay best;
    best.SetRefMol(refmol);

    cout << "Ref Title: " << refmol.GetTitle() << " ";
    cout << "Num Confs: " << refmol.NumConfs() << endl << endl;

    OEBestOverlayScore *score;
    OEMol fitmol;
    while (OEReadMolecule(fitfs, fitmol))
    {
        cout << "Fit Title: " << fitmol.GetTitle() << " ";
        cout << "Num Confs: " << fitmol.NumConfs() << endl;

        unsigned int resCount = 0;
        OEIter<OEBestOverlayResults> resiter = best.Overlay(fitmol);
        for (;resiter;++resiter)
        {
            OEIter<OEBestOverlayScore> scoreiter = resiter->GetScores();
            for (;scoreiter;++scoreiter)
            {
                score = scoreiter;
                cout << "FitConfIdx: " << score->fitconfidx << " ";
                cout << "RefConfIdx: " << score->refconfidx << " ";
                cout << "Tanimoto: " << score->tanimoto << endl;
                ++resCount;
            }
        }
        cout << resCount << " results returned" << endl << endl;
    }
    return 0;
}
```

But in most cases, one does not want or need all the results. Most times, the single best overlap for each conformer-conformer pair is desired. The next example shows the `OESortOverlayScores` function used to turn the double iterator as shown in the example above into a single iterator of `OEBestOverlayScore`. Note that the third argument

is a functor used to sort the list, such that in this next example, we get one `OEBestOverlayScore` for each pair of conformers and they are returned in Tanimoto order.

Listing 7: Getting all the best scores from OEBestOverlay.

```

/*****
Copyright (C) 2005-2010 OpenEye Scientific Software, Inc.
*****/
#include "openeye.h"

#include "oeplatform.h"
#include "oesystem.h"
#include "oechem.h"
#include "oeshape.h"

using namespace OEPlatform;
using namespace OESystem;
using namespace OEChem;
using namespace OEShape;
using namespace std;

int main(int argc, char *argv[])
{
    if (argc!=3)
        OEThrow.Usage("bestoverlay2 <reffile> <fitfile>");

    oemolistream reffs(argv[1]);
    oemolistream fitfs(argv[2]);

    OEMol refmol;
    OEReadMolecule(reffs, refmol);

    OEBestOverlay best;
    best.SetRefMol(refmol);

    cout << "Ref Title: " << refmol.GetTitle() << " ";
    cout << "Num Confs: " << refmol.NumConfs() << endl << endl;

    OEBestOverlayScore *score;
    OEMol fitmol;
    while (OEReadMolecule(fitfs, fitmol))
    {
        cout << "Fit Title: " << fitmol.GetTitle() << " ";
        cout << "Num Confs: " << fitmol.NumConfs() << endl;

        unsigned int resCount=0;
        OEIter<OEBestOverlayResults> resiter = best.Overlay(fitmol);
        OEIter<OEBestOverlayScore> scoreiter;
        OESortOverlayScores(scoreiter, resiter, OEHighestTanimoto());
        for (;scoreiter;++scoreiter)
        {
            score = scoreiter;
            cout << "FitConfIdx: " << score->fitconfidx << " ";
            cout << "RefConfIdx: " << score->refconfidx << " ";
            cout << "Tanimoto: " << score->tanimoto << endl;
            ++resCount;
        }
    }
}

```

```
    }  
    cout << resCount << " results returned" << endl << endl;  
  }  
  return 0;  
}
```

The next example is a slight variation on the previous. It adds a user parameter for the number of results to keep. Since the results are sorted by Tanimoto, keeping the first 5, keeps the best 5.

Listing 8: Keeping a few top scores from OEBestOverlay.

```
/*  
Copyright (C) 2005-2010 OpenEye Scientific Software, Inc.  
*/  
#include "openeye.h"  
  
#include "oeplatform.h"  
#include "oesystem.h"  
#include "oechem.h"  
#include "oeshape.h"  
  
using namespace OEPlatform;  
using namespace OESystem;  
using namespace OEChem;  
using namespace OEShape;  
using namespace std;  
  
int main(int argc, char *argv[])  
{  
  if (argc!=4)  
    OEThrow.Usage("bestoverlay3 <reffile> <fitfile> <keepsizesize>");  
  
  oemolistream reffs(argv[1]);  
  oemolistream fitfs(argv[2]);  
  unsigned int keepsize = atoi(argv[3]);  
  
  OEMol refmol;  
  OEReadMolecule(reffs, refmol);  
  
  OEBestOverlay best;  
  best.SetRefMol(refmol);  
  
  cout << "Ref Title: " << refmol.GetTitle() << " ";  
  cout << "Num Confs: " << refmol.NumConfs() << endl << endl;  
  
  OEBestOverlayScore *score;  
  OEMol fitmol;  
  while (OEReadMolecule(fitfs, fitmol))  
  {  
    cout << "Fit Title: " << fitmol.GetTitle() << " ";  
    cout << "Num Confs: " << fitmol.NumConfs() << endl;  
  
    unsigned int resCount=0;  
    OEIter<OEBestOverlayResults> resiter = best.Overlay(fitmol);  
    OEIter<OEBestOverlayScore> scoreiter;  
    OESortOverlayScores(scoreiter, resiter, OEHighestTanimoto());  
  }  
}
```

```

for (;scoreiter;++scoreiter)
{
    score = scoreiter;
    cout << "FitConfIdx: " << score->fitconfidx << " ";
    cout << "RefConfIdx: " << score->refconfidx << " ";
    cout << "Tanimoto: " << score->tanimoto << endl;
    ++resCount;
    if (resCount==keepsizes)
        break;
}
cout << resCount << " results returned" << endl << endl;
}
return 0;
}

```

3.3.3 Saving aligned structures

OEBestOverlay does not actually move the fit molecule. Part of OEBestOverlayScore is the rotation matrix and translation matrix necessary to move the fit molecule into the final overlap position. (N.B. the rotation matrix is left-multiplied so can be regarded as left-handed.) It is up to the user to apply these transformations. The OpenEye standard is rotation then translation, but as a convenience, OEBestOverlayScore has `OEBestOverlayScore::Transform` method that will apply the transforms in the proper order. This next example expands from the previous. Now, not only can we choose the number of overlays to keep, we are also going to align each fit conformer to the ref conformer it was overlaid on, and then write the pair to an output file. If SD or OEB is used as the output file type, then scores will also be stored in SD tags.

Listing 9: Writing aligned structures from OEBestOverlay.

```

/*****
Copyright (C) 2005-2010 OpenEye Scientific Software, Inc.
*****/
#include "openeye.h"

#include "oeplatform.h"
#include "oesystem.h"
#include "oechem.h"
#include "oeshape.h"

using namespace OEPlatform;
using namespace OESystem;
using namespace OEChem;
using namespace OEShape;
using namespace std;

int main(int argc, char *argv[])
{
    if (argc!=5)
        OEThrow.Usage("bestoverlay4 <reffile> <fitfile> <output.sdf> <keepsizes>");

    oemolistream reffs(argv[1]);
    oemolistream fitfs(argv[2]);
    oemolistream outfs(argv[3]);
    unsigned int keepsizes = atoi(argv[4]);
}

```

```
OEMol refmol;
OEReadMolecule(reffs, refmol);

OEBestOverlay best;
best.SetRefMol(refmol);

cout << "Ref Title: " << refmol.GetTitle() << " ";
cout << "Num Confs: " << refmol.NumConfs() << endl << endl;

OEBestOverlayScore *score;
OEMol fitmol;
while (OEReadMolecule(fitfs, fitmol))
{
    cout << "Fit Title: " << fitmol.GetTitle() << " ";
    cout << "Num Confs: " << fitmol.NumConfs() << endl;

    unsigned int resCount=0;
    OEIter<OEBestOverlayResults> resiter = best.Overlay(fitmol);
    OEIter<OEBestOverlayScore> scoreiter;
    OESortOverlayScores(scoreiter, resiter, OEHighestTanimoto());
    for (;scoreiter;++scoreiter)
    {
        score = scoreiter;
        OEGraphMol outmol(*fitmol.GetConf(OEHasConfIdx(score->fitconfidx)));
        score->Transform(outmol);

        char buffer[12];
        sprintf(buffer, "%-d", score->refconfidx);
        OESetSDData(outmol, "RefConfIdx", buffer);
        sprintf(buffer, "%-6.3f", score->tanimoto);
        OESetSDData(outmol, "ShapeTanimoto", buffer);

        OEWiteMolecule(outfs, *refmol.GetConf(OEHasConfIdx(score->refconfidx)));
        OEWiteMolecule(outfs, outmol);

        ++resCount;
        if (resCount==keepsz)
            break;
    }
    cout << resCount << " results returned" << endl << endl;
}
return 0;
}
```

3.3.4 Using color along with shape

In order to maximize chemical similarity along with shape, an `OECOLORForceField` can be added to `OEBestOverlay`. Once added, color scores will be calculated along with shape overlap scores. By default, the color force field only appears as a post-optimization scoring function. That is, after the shape overlap is maximized, color scores are calculated.

Color force can, however, also be used as part of the optimization. Color optimization is turned on by the `OEBestOverlay::SetColorOptimize` member function.

Listing 10: Using color with OEBestOverlay.

```

/*****
Copyright (C) 2005-2010 OpenEye Scientific Software, Inc.
*****/
#include "openeye.h"

#include "oeplatform.h"
#include "oesystem.h"
#include "oechem.h"
#include "oeshape.h"

using namespace OEPlatform;
using namespace OESystem;
using namespace OEChem;
using namespace OEShape;
using namespace std;

int main(int argc, char *argv[])
{
    if (argc!=5)
        OEThrow.Usage("bestoverlay-color <reffile> <fitfile> <output.sdf> <keepsizesize>");

    oemolistream reffs(argv[1]);
    oemolistream fitfs(argv[2]);
    oemolistream outfs(argv[3]);
    unsigned int keepsize = atoi(argv[4]);

    OEMol refmol;
    OEReadMolecule(reffs, refmol);

    OEBestOverlay best;
    best.SetRefMol(refmol);
    best.SetColorForceField(OEColorFFType::ImplicitMillsDean);
    best.SetColorOptimize(true);

    cout << "Ref Title: " << refmol.GetTitle() << " ";
    cout << "Num Confs: " << refmol.NumConfs() << " ";
    cout << "Self color: " << best.GetRefSelfColor() << endl << endl;

    OEBestOverlayScore *score;
    OEMol fitmol;
    while (OEReadMolecule(fitfs, fitmol))
    {
        cout << "Fit Title: " << fitmol.GetTitle() << " ";
        cout << "Num Confs: " << fitmol.NumConfs() << endl;

        unsigned int resCount=0;
        OEIter<OEBestOverlayResults> resiter = best.Overlay(fitmol);
        OEIter<OEBestOverlayScore> scoreiter;
        OESortOverlayScores(scoreiter, resiter, OEHighestTanimotoCombo());
        for (;scoreiter;++scoreiter)
        {
            score = scoreiter;
            OEGraphMol outmol(*fitmol.GetConf(OEHasConfIdx(score->fitconfidx)));
            score->Transform(outmol);
        }
    }
}

```

```
    char buffer[12];
    sprintf(buffer, "%-d", score->refconfidx);
    OESetSDData(outmol, "RefConfIdx", buffer);
    sprintf(buffer, "%-6.3f", score->GetTanimotoCombo());
    OESetSDData(outmol, "TanimotoCombo", buffer);
    sprintf(buffer, "%-6.3f", score->GetShapeTanimoto());
    OESetSDData(outmol, "ShapeTanimoto", buffer);
    sprintf(buffer, "%-6.3f", score->GetColorTanimoto());
    OESetSDData(outmol, "ColorTanimoto", buffer);

    OEWriteMolecule(outfs, *refmol.GetConf(OEHasConfIdx(score->refconfidx)));
    OEWriteMolecule(outfs, outmol);

    ++resCount;
    if (resCount==keepsize)
        break;
}
cout << resCount << " results returned" << endl << endl;
}
return 0;
}
```

3.4 Calculating NxN scores

There are times when you want to have all the pair-wise scores among a set of molecules. Maintaining this 2D matrix of scores is more complicated than the single set of scores from the *ref vs. set of fit molecules* examples shown above.

This example takes a set of molecules and performs all the pair-wise shape optimizations with `OEBestOverlay`. It outputs a *spreadsheet* of the scores.

Listing 11: Calculating NxN scores.

```
/*
Copyright (C) 2005,2006,2007,2008 OpenEye Scientific Software, Inc.
*/
#include "openeye.h"

#include "oeplatform.h"
#include "oesystem.h"
#include "oechem.h"
#include "oeshape.h"

using namespace OEPlatform;
using namespace OESystem;
using namespace OEChem;
using namespace OEShape;
using namespace std;

void oneConf(const OEConfBase &conf, const char *filename,
            oeofstream &csvfile)
{
    char buffer[8];
    csvfile << conf.GetTitle() << "_" << conf.GetIdx();
}
```

```

OEMol refmol(conf);
OEBestOverlay best;
best.SetRefMol(refmol);

oemolistream bfs(filename);
OEMol fitmol;
OEIter<OEBestOverlayScore> scoreiter;
while (OEReadMolecule(bfs, fitmol))
{
    OEIter<OEBestOverlayResults> resiter = best.Overlay(fitmol);
    OESortOverlayScores(scoreiter, resiter, OEHighestTanimoto(), 1, true);
    for (;scoreiter;++scoreiter)
    {
        sprintf(buffer, "%.2f", scoreiter->tanimoto);
        csvfile << "," << buffer;
    }
}
csvfile << oeendl;
}

void genHeader(const char *filename, oeofstream &csvfile)
{
    csvfile << "name";
    oemolistream ifs(filename);
    OEMol mol;
    while (OEReadMolecule(ifs, mol))
    {
        for (OEIter<OEConfBase> citer=mol.GetConfs();citer;++citer)
        {
            csvfile << "," << citer->GetTitle() << "_" << citer->GetIdx();
        }
    }
    csvfile << oeendl;
}

int main(int argc, char *argv[])
{
    if (argc!=3)
        OETHrow.Usage("nxnshape <infile> <file.csv>");

    oeofstream csvfile(argv[2]);
    genHeader(argv[1], csvfile);

    oemolistream afs(argv[1]);
    OEMol molA;
    while (OEReadMolecule(afs, molA))
    {
        oeerr << molA.GetTitle() << oeendl;
        for (OEIter<OEConfBase> citer=molA.GetConfs();citer;++citer)
        {
            oneConf(citer, argv[1], csvfile);
        }
    }
    return 0;
}

```

3.5 Calculating shape characteristics

While most functionality in the Shape Toolkit involves comparison of pairs of molecules, there are a few fundamental properties that can be calculated for a single molecule. All of these calculations are done using the same basic Gaussian description of a molecule as described above.

The simplest property to calculate is volume, using `OECalcVolume`.

In addition to simple volume calculations, the steric multipoles of a molecule can also be calculated. See section `OECalcShapeMultipoles`.

This next example demonstrates the calculation of volume and shape multipoles.

Listing 12: Calculating shape properties.

```
/*  
Copyright (C) 2005,2006,2007,2008 OpenEye Scientific Software, Inc.  
*/  
#include "openeye.h"  
  
#include "oeplatform.h"  
#include "oesystem.h"  
#include "oechem.h"  
#include "oeshape.h"  
  
using namespace OEPlatform;  
using namespace OESystem;  
using namespace OEChem;  
using namespace OEShape;  
using namespace std;  
  
int main(int argc, char *argv[])  
{  
    if (argc!=2)  
        OEThrow.Usage("shapeprops <infile>");  
  
    oemolistream ifs(argv[1]);  
    oemolostream ofs(argv[2]);  
  
    OEGraphMol mol;  
    while (OEReadMolecule(ifs, mol))  
    {  
        OEThrow.Info("          Title: %s", mol.GetTitle());  
        OEThrow.Info("          Volume: %8.2f", OECalcVolume(mol));  
        OEThrow.Info("Volume: (without H): %8.2f", OECalcVolume(mol, false));  
  
        float smult[14];  
        OECalcShapeMultipoles(mol, smult);  
  
        OEThrow.Info("  Steric multipoles:");  
        OEThrow.Info("          monopole: %8.2f", smult[0]);  
        OEThrow.Info("          quadrupoles: %8.2f %8.2f %8.2f",  
            smult[1], smult[2], smult[3]);  
        OEThrow.Info("          octopoles:");  
        OEThrow.Info("          xxx: %8.2f  yyy: %8.2f  zzz: %8.2f",  
            smult[4], smult[5], smult[6]);  
    }  
}
```

```
OETrow.Info("          xxy: %8.2f  xxz: %8.2f  yyx: %8.2f",
            smult[7], smult[8], smult[9]);
OETrow.Info("          yyz: %8.2f  zzx: %8.2f  zzy: %8.2f",
            smult[10], smult[11], smult[12]);
OETrow.Info("          xyz: %8.2f", smult[13]);

OETrow.Info("");

}

return 0;
}
```


4.1 OEShape Classes

4.1.1 OEBestOverlay

`class OEBestOverlay`

This class is used to optimize the overlap between 2 molecules. Both molecules can contain one or more conformers. The reference molecule is held rigid and the fit molecule orientation is calculated to maximize overlap. Each conformer of the fit molecule is optimized against each conformer of the reference molecule, resulting in a large number of results that can be returned. The `OEBestOverlayResults` and `OEBestOverlayScore` classes are used to make handling this large amount of results easier.

Additionally, the reference or fit shape can be a grid instead of a molecule.

Since an `OEBestOverlay` optimization can result in numerous final results, one for each starting position, the `OEBestOverlayResults` class is a container to hold all the results for a fit conformer against a reference conformer. This class is essentially a container for one or more `OEBestOverlayScore` instances. The actual number of scores depends on the `OEBOOrientation` used in the `OEBestOverlay`.

For `OEBOOrientation::Inertial`, there will be 4, 8 or 20 scores per ref-fit conformer pair, depending on the symmetry of each conformer. For `OEBOOrientation::AsIs`, there will be just 1 `OEBestOverlayScore` inside each `OEBestOverlayResults` instance. And for `OEBOOrientation::Random`, there will be 1 `OEBestOverlayScore` for each of the `N` random starts.

When dealing with the results from `OEBestOverlay`, the user can either use a double loop, *i.e.* loop over all the `OEBestOverlayResults` instances for each ref-fit conformer pair then loop over each `OEBestOverlayScore` inside each `OEBestOverlayResults` instance. In this case, the results also come out in conformer order.

```
OEIter<OEBestOverlayResults> resiter = best.Overlay(fitmol);
for (;resiter;++resiter)
{
    OEIter<OEBestOverlayScore> scoreiter = resiter->GetScores();
    for (;scoreiter;++scoreiter)
    {
        // do something with score
    }
}
```

Or, there is a free function `OESortOverlayScores`, that takes an iterator of `OEBestOverlayResults` and returns a single, sorted iterator of `OEBestOverlayScores` that can be used in a single loop.

```
OEIter<OEBestOverlayResults> resiter = best.Overlay(fitmol);
OEIter<OEBestOverlayScore> scoreiter;
OESortOverlayScores(scoreiter, resiter, OEHighestTanimotoCombo());
for (;scoreiter;++scoreiter)
{
    // do something with score
}
```

Constructors

```
OEBestOverlay()
OEBestOverlay(const OEBestOverlay &)
OEBestOverlay(const OEChem::OEMCMolBase &refmol)
OEBestOverlay(const OESystem::OEScalarGrid &refgrid, float interpolate=0.5f)
```

An empty OEBestOverlay class instance can be created, or a new instance can take the reference molecule or grid as an argument. Note that by default, grids that have a resolution larger than 0.5 Å will be interpolated to that resolution.

operator=

```
OEBestOverlay &operator=(const OEBestOverlay &)
```

ClearColorForceField

```
void ClearColorForceField()
```

Clear out any color force field. No color scores will be calculated.

ClearUserStarts

```
void ClearUserStarts()
```

Set the number of User Starts back to zero.

GetAllColor

```
bool GetAllColor() const
```

GetCarbonRadius

```
float GetCarbonRadius() const
```

Return the current value for the carbon radius approximation.

GetInertialAxialDivisions

`unsigned int` GetInertialAxialDivisions() `const`

GetInitialOrientation

`unsigned int` GetInitialOrientation() `const`

Get the current value for initial orientation. Possible values are from the `OEB0Orientation` namespace.

GetMaxRandomTranslation

`float` GetMaxRandomTranslation() `const`

Get the value in Å of maximum random translation.

GetMethod

`unsigned int` GetMethod() `const`

Return the current value of overlap method.

GetMinimizeType

`unsigned int` GetMinimizeType() `const`

Get the current value for minimize type. Possible values are from the `OEB0MinType` namespace.

GetNumRandomStarts

`unsigned int` GetNumRandomStarts() `const`

Get the current number of random starts.

GetNumUserStarts

`unsigned int` GetNumUserStarts() `const`

Get the number of user starts currently set. Note that these are not actually used unless `OEB0Orientatin::UserInertialStarts` is also set.

GetRadiiApproximation

`unsigned int` GetRadiiApproximation() `const`

Return the current value of the radii approximation.

GetRandomSeed

```
unsigned int GetRandomSeed() const
```

Get the current value of the random seed.

GetRefGrid

```
const OESystem::OEScalarGrid *GetRefGrid() const
```

GetRefMol

```
const OEChem::OEMCMolBase *GetRefMol() const
```

GetRefSelfColor

```
float GetRefSelfColor()
```

Return the self color score of the reference molecule.

GetRefSymmetry

```
unsigned int GetRefSymmetry(unsigned int confIdx) const
```

GetRepresentationLevel

```
unsigned int GetRepresentationLevel() const
```

Return the current representation level.

GetSymmetryThreshold

```
float GetSymmetryThreshold() const
```

GetUseHydrogens

```
bool GetUseHydrogens() const
```

Return the status of hydrogen use in OEBestOverlay.

GetUserStarts

```
bool GetUserStarts(float *xyz) const
```

Extract the coordinates of the user starts. xyz should be sized to 3 * GetNumUserStarts().

Overlay

```
OESystem::OEIterBase<OEBestOverlayResults> *
  Overlay(const OEChem::OEMCMolBase &fitmol)
OESystem::OEIterBase<OEBestOverlayResults> *
  Overlay(const OESystem::OEScalarGrid &fitgrid)
```

Perform the calculation and return the an iterator of the results.

SetAllColor

```
void SetAllColor(bool state)
```

SetCarbonRadius

```
void SetCarbonRadius(float cradius)
```

Set the radius to use when using `OEOverlapRadii::Carbon`. By default this is set to 1.7 Å.

SetColorForceField

```
bool SetColorForceField(unsigned int type)
bool SetColorForceField(OEPlatform::oeistream &is)
bool SetColorForceField(const std::string &filename)
bool SetColorForceField(const OECOLORForceField &cff)
```

Set the color force field to be used. Once set, color scores will be calculated and included in the results. Color gradients will not be included in the optimization unless specifically set using `SetColorOptimize(true)`.

SetColorOptimize

```
void SetColorOptimize(bool state)
```

Add color gradients to shape gradients in the optimization. Has no effect unless a color force field is also set via `OEBestOverlay::SetColorForceField`.

SetInertialAxialDivisions

```
void SetInertialAxialDivisions(unsigned int divisions)
```

SetInitialOrientation

void SetInitialOrientation(**unsigned int** orient)

Determines the initial orientation (starting position) for each optimization. The default is `OEBOrientation::Inertial`. Alternatives are defined in the `OEBOrientation` namespace.

SetMaxRandomTranslation

void SetMaxRandomTranslation(**float** trans)

If using random starts, this set the maximum distance (in Å) that the center of mass of the fit molecule will be moved away from the center calculated for inertial frame alignment.

SetMethod

void SetMethod(**unsigned int** method)

Set the method used to calculate overlap. The default for `OEBestOverlay` is `OEOverlapMethod::Grid`. Alternatives are defined in the `OEOverlapMethod` namespace.

SetMinimizeType

void SetMinimizeType(**unsigned int** type)

Set the score to use in the optimization. Options are in the `OEBOMinType` namespace.

SetNumRandomStarts

void SetNumRandomStarts(**unsigned int** n)

If `SetInitialOrientation` is set to `OEBOrientation::Random`, this method sets the number of random starting positions that will be used.

SetRadiiApproximation

void SetRadiiApproximation(**unsigned int** type)

Set the radius approximation used to calculate overlap. The default for `OEBestOverlay` is `OEOverlapRadii::Carbon`. Alternatives are defined in the `OEOverlapRadii` namespace.

SetRandomSeed

void SetRandomSeed(**unsigned int** seed)

Set a random seed value to allow reproducible random searches.

SetRefGrid

```
bool SetRefGrid(const OESystem::OEScalarGrid &refgrid, float interpolate=0.5f)
```

Set a reference grid for the calculation. OEBestOverlay makes an internal copy. Pre-existing reference molecules or grids are replaced.

SetRefMol

```
bool SetRefMol(const OEChem::OEMCMolBase &refmol)
```

Set a reference molecule for the calculation. OEBestOverlay makes an internal copy. Pre-existing reference molecules or grids are replaced.

SetRepresentationLevel

```
void SetRepresentationLevel(unsigned int type)
```

Set the representation level for the gaussians in OEBestOverlay. The default is `OEOverlapRepresentation::Atomic`. Alternatives are defined in the `OEOverlapRepresentation` namespace.

SetSymmetryThreshold

```
void SetSymmetryThreshold(float threshold)
```

SetUseHydrogens

```
void SetUseHydrogens(bool state)
```

Boolean to determine whether hydrogens are included in the shape calculation. By default this is false and hydrogens are ignored.

SetUserStarts

```
void SetUserStarts(const float *xyz, unsigned int nstarts)
```

Set a set of 3D coordinates for doing separate inertial starts. `xyz` should be $3 * nstarts$. These are not used unless `OEBOrientation::UserInertialStarts` is also set.

4.1.2 OEBestOverlayResults

```
class OEBestOverlayResults
```

Constructors

```
OEBestOverlayResults()  
OEBestOverlayResults(const OEBestOverlayResults &rhs)
```

Default and copy constructors.

operator=

```
OEBestOverlayResults &operator=(const OEBestOverlayResults &rhs)
```

AddScore

```
bool AddScore(const OEBestOverlayScore &sc)
```

Clear

```
bool Clear()
```

GetScores

```
OESystem::OEIterBase<OEBestOverlayScore> *GetScores() const  
OESystem::OEIterBase<OEBestOverlayScore> *  
  GetScores(const OESystem::OEBinaryPredicate<OEBestOverlayScore,  
    OEBestOverlayScore> &sorter, int nbest=1) const
```

4.1.3 OEBestOverlayScore

```
class OEBestOverlayScore
```

This class represents *OEBestOverlayScore*.

Constructors

```
OEBestOverlayScore()  
OEBestOverlayScore(const OEBestOverlayScore &rhs)
```

Default and copy constructors.

operator=

```
OEBestOverlayScore &operator=(const OEBestOverlayScore &rhs)
```

GetColorTanimoto

`float` GetColorTanimoto() `const`

GetColorTversky

`float` GetColorTversky(`float` alpha=0.95f, `float` beta=0.05f) `const`

GetComboScore

`float` GetComboScore() `const`

GetFitColorTversky

`float` GetFitColorTversky() `const`

GetFitTversky

`float` GetFitTversky() `const`

GetFitTverskyCombo

`float` GetFitTverskyCombo() `const`

GetRefColorTversky

`float` GetRefColorTversky() `const`

GetRefTversky

`float` GetRefTversky() `const`

GetRefTverskyCombo

`float` GetRefTverskyCombo() `const`

GetRotMatrix

```
void GetRotMatrix(float *rmat) const
```

GetTanimotoCombo

```
float GetTanimotoCombo() const
```

GetTversky

```
float GetTversky(float alpha=0.95f, float beta=0.05f) const
```

Transform

```
bool Transform(OEChem::OEMolBase &mol) const  
bool Transform(OEChem::OEConfBase &mol) const
```

GetTranslation

```
void GetTranslation(float *trans) const
```

4.1.4 OEColorForceField

```
class OEColorForceField
```

This class represents *OEColorForceField*.

Constructors

```
OEColorForceField()  
OEColorForceField(const OEColorForceField &rhs)  
OEColorForceField(const OEColorForceFieldImpl *impl)
```

Default and copy constructors.

operator=

```
OEColorForceField &operator=(const OEColorForceField &rhs)
```

AddColorer

```
bool AddColorer(unsigned int type, const char *smarts)
bool AddColorer(unsigned int type, const std::string &smarts)
```

AddInteraction

```
bool AddInteraction(unsigned int type1, unsigned int type2,
                   const std::string &interaction_type, float weight,
                   float range)
```

AddType

```
unsigned int AddType(const char *name)
unsigned int AddType(const std::string &name)
```

Clear

```
void Clear()
```

ClearInteractions

```
void ClearInteractions()
```

GetTitle

```
const char *GetTitle() const
```

GetType

```
unsigned int GetType(std::string k) const
```

GetTypeName

```
std::string GetTypeName(unsigned int type) const
```

Init

```
bool Init(unsigned int cffType)
bool Init(OEPlatform::oeistream &is, bool verbose=false)
bool Init(const std::string &filename, bool verbose=false)
```

Ready

```
bool Ready() const
```

SetTitle

```
bool SetTitle(const char *title)
```

Write

```
bool Write(OEPlatform::oeostream &os)
```

4.1.5 OEColorOverlap

```
class OEColorOverlap
```

This class represents *OEColorOverlap*.

Constructors

```
OEColorOverlap()  
OEColorOverlap(const OEColorOverlap &rhs)
```

Default and copy constructors.

operator=

```
OEColorOverlap &operator=(const OEColorOverlap &rhs)
```

ColorScore

```
bool ColorScore(OEColorResults &res)  
bool ColorScore(const OEChem::OEMolBase &fitmol, OEColorResults &res)
```

GetAllColor

```
bool GetAllColor() const
```

GetSelfColor

```
float GetSelfColor() const
```

SetAllColor

```
void SetAllColor(bool state)
```

SetColorForceField

```
bool SetColorForceField(unsigned int cffType)
bool SetColorForceField(OEPlatform::oeistream &is)
bool SetColorForceField(const std::string &filename)
bool SetColorForceField(const OECColorForceField &cff)
```

SetFitMol

```
bool SetFitMol(const OEChem::OEMolBase &fitmol)
```

SetRefMol

```
bool SetRefMol(const OEChem::OEMolBase &refmol)
```

4.1.6 OECColorResults

```
struct OECColorResults
```

This class represents *OECColorResults*.

Constructors

```
OECColorResults()
```

Default and copy constructors.

GetFitTversky

```
float GetFitTversky() const
```

GetRefTversky

```
float GetRefTversky() const
```

GetTanimoto

```
float GetTanimoto() const
```

GetTversky

```
float GetTversky(float alpha, float beta) const
```

4.1.7 OEHighestColorTanimoto

```
struct OEHighestColorTanimoto : public OESystem::OEBinaryPredicate<OEBestOverlayScore, OEBestOverlayScore>
```

This class represents *OEHighestColorTanimoto*.

operator()

```
bool operator() (const OEBestOverlayScore &s1,  
                const OEBestOverlayScore &s2) const
```

CreateCopy

```
OESystem::OEBinaryFunction<OEBestOverlayScore , OEBestOverlayScore , bool> *  
CreateCopy() const
```

4.1.8 OEHighestComboScore

```
struct OEHighestComboScore : public OESystem::OEBinaryPredicate<OEBestOverlayScore, OEBestOverlayScore>
```

This class represents *OEHighestComboScore*.

operator()

```
bool operator() (const OEBestOverlayScore &s1,  
                const OEBestOverlayScore &s2) const
```

CreateCopy

```
OESystem::OEBinaryFunction<OEBestOverlayScore , OEBestOverlayScore , bool> *  
CreateCopy() const
```

4.1.9 OEHighestFitColorTversky

```
struct OEHighestFitColorTversky : public OESystem::OEBinaryPredicate<OEBestOverlayScore, OEBestOverlayScore>
```

This class represents *OEHighestFitColorTversky*.

operator()

```
bool operator() (const OEBestOverlayScore &s1,  
                const OEBestOverlayScore &s2) const
```

CreateCopy

```
OESystem::OEBinaryFunction<OEBestOverlayScore , OEBestOverlayScore , bool> *  
  CreateCopy() const
```

4.1.10 OEHighestFitTversky

```
struct OEHighestFitTversky : public OESystem::OEBinaryPredicate<OEBestOverlayScore, OEBestOverlayScore>
```

This class represents *OEHighestFitTversky*.

operator()

```
bool operator() (const OEBestOverlayScore &s1,  
                const OEBestOverlayScore &s2) const
```

CreateCopy

```
OESystem::OEBinaryFunction<OEBestOverlayScore , OEBestOverlayScore , bool> *  
  CreateCopy() const
```

4.1.11 OEHighestFitTverskyCombo

```
struct OEHighestFitTverskyCombo : public OESystem::OEBinaryPredicate<OEBestOverlayScore, OEBestOverlayScore>
```

This class represents *OEHighestFitTverskyCombo*.

operator()

```
bool operator() (const OEBestOverlayScore &s1,  
                const OEBestOverlayScore &s2) const
```

CreateCopy

```
OESystem::OEBinaryFunction<OEBestOverlayScore , OEBestOverlayScore , bool> *  
    CreateCopy() const
```

4.1.12 OEHighestOverlap

```
struct OEHighestOverlap : public OESystem::OEBinaryPredicate<OEBestOverlayScore, OEBestOverlayScore>
```

This class represents *OEHighestOverlap*.

operator()

```
bool operator() (const OEBestOverlayScore &s1,  
                const OEBestOverlayScore &s2) const
```

CreateCopy

```
OESystem::OEBinaryFunction<OEBestOverlayScore , OEBestOverlayScore , bool> *  
    CreateCopy() const
```

4.1.13 OEHighestRefColorTversky

```
struct OEHighestRefColorTversky : public OESystem::OEBinaryPredicate<OEBestOverlayScore, OEBestOverlayScore>
```

This class represents *OEHighestRefColorTversky*.

operator()

```
bool operator() (const OEBestOverlayScore &s1,  
                const OEBestOverlayScore &s2) const
```

CreateCopy

```
OESystem::OEBinaryFunction<OEBestOverlayScore , OEBestOverlayScore , bool> *  
    CreateCopy() const
```

4.1.14 OEHighestRefTversky

```
struct OEHighestRefTversky : public OESystem::OEBinaryPredicate<OEBestOverlayScore, OEBestOverlayScore>
```

This class represents *OEHighestRefTversky*.

operator()

```
bool operator() (const OEBestOverlayScore &s1,
                const OEBestOverlayScore &s2) const
```

CreateCopy

```
OESystem::OEBinaryFunction<OEBestOverlayScore , OEBestOverlayScore , bool> *
  CreateCopy() const
```

4.1.15 OEHighestRefTverskyCombo

```
struct OEHighestRefTverskyCombo : public OESystem::OEBinaryPredicate<OEBestOverlayScore, OEBestOverlayScore>
```

This class represents *OEHighestRefTverskyCombo*.

operator()

```
bool operator() (const OEBestOverlayScore &s1,
                const OEBestOverlayScore &s2) const
```

CreateCopy

```
OESystem::OEBinaryFunction<OEBestOverlayScore , OEBestOverlayScore , bool> *
  CreateCopy() const
```

4.1.16 OEHighestScaledColor

```
struct OEHighestScaledColor : public OESystem::OEBinaryPredicate<OEBestOverlayScore, OEBestOverlayScore>
```

This class represents *OEHighestScaledColor*.

operator()

```
bool operator() (const OEBestOverlayScore &s1,
                const OEBestOverlayScore &s2) const
```

CreateCopy

```
OESystem::OEBinaryFunction<OEBestOverlayScore , OEBestOverlayScore , bool> *
  CreateCopy() const
```

4.1.17 OEHighestTanimoto

```
struct OEHighestTanimoto : public OESystem::OEBinaryPredicate<OEBestOverlayScore, OEBestOverlayScore>
```

This class represents *OEHighestTanimoto*.

operator()

```
bool operator() (const OEBestOverlayScore &s1,  
                const OEBestOverlayScore &s2) const
```

CreateCopy

```
OESystem::OEBinaryFunction<OEBestOverlayScore , OEBestOverlayScore , bool> *  
    CreateCopy() const
```

4.1.18 OEHighestTanimotoCombo

```
struct OEHighestTanimotoCombo : public OESystem::OEBinaryPredicate<OEBestOverlayScore, OEBestOverlayScore>
```

This class represents *OEHighestTanimotoCombo*.

operator()

```
bool operator() (const OEBestOverlayScore &s1,  
                const OEBestOverlayScore &s2) const
```

CreateCopy

```
OESystem::OEBinaryFunction<OEBestOverlayScore , OEBestOverlayScore , bool> *  
    CreateCopy() const
```

4.1.19 OEIsColorAtomPred

```
class OEIsColorAtomPred : public OESystem::OEUnaryPredicate<OEChem::OEAtomBase>
```

This class represents *OEIsColorAtomPred*.

Constructors

```
OEIsColorAtomPred()
```

Default and copy constructors.

operator()

```
bool operator() (const OEChem::OEAtomBase &atom) const
```

CreateCopy

```
OESystem::OEUnaryFunction<OEChem::OEAtomBase, bool> *CreateCopy() const
```

4.1.20 OEOverlap

```
class OEOverlap
```

OEOverlap calculates the static shape overlap between a reference molecule or grid and a fit molecule or grid. Note that this does not move the fit molecule(grid) nor does it optimize the overlap. It simply calculates the score for the provided orientation.

Constructors

```
OEOverlap()
OEOverlap(const OEOverlap &rhs)
OEOverlap(const OEChem::OEMolBase &refmol)
OEOverlap(const OESystem::OEScalarGrid &refmol)
```

Default and copy constructors.

operator=

```
OEOverlap &operator=(const OEOverlap &rhs)
```

Assignment operator. The contents of the passed OEOverlap reference are copied into the current OEOverlap instance.

GetCarbonRadius

```
float GetCarbonRadius() const
```

Return the current value for the carbon radius approximation.

GetHandle

```
int GetHandle() const
```

Return the current handle. This is an implementation detail and not used by end-user programs.

GetMethod

```
unsigned int GetMethod() const
```

Return the current value of overlap method.

GetRadiiApproximation

```
unsigned int GetRadiiApproximation() const
```

Return the current value of the radii approximation.

GetRepresentationLevel

```
unsigned int GetRepresentationLevel() const
```

Return the current representation level.

GetUseHydrogens

```
bool GetUseHydrogens() const
```

Return the status of hydrogen use in OEOverlap.

Overlap

```
bool Overlap(OEOverlapResults &res, float *atomOverlaps=0)
bool Overlap(const OESystem::OEScalarGrid &fitgrid, OEOverlapResults &res)
bool Overlap(const OEChem::OEMolBase &fitmol, OEOverlapResults &res,
             float *atomOverlaps=0)
```

Calculate the overlap of the passed in fit molecule or fit grid and place the results into the passed instance of OEOverlapResults.

SetCarbonRadius

```
bool SetCarbonRadius(float cradius)
```

Set the radius to use when using Carbon. By default this is set to 1.7 Angstroms. See `OEBestOverlay::GetCarbonRadius`.

SetFitGrid

```
bool SetFitGrid(const OESystem::OEScalarGrid &fitgrid)
```

Set grid to be used as fit object.

SetFitMol

```
bool SetFitMol(const OEChem::OEMolBase &fitmol)
```

Set molecule to be used as fit object.

SetMethod

```
bool SetMethod(unsigned int m)
```

Set the method used to calculate overlap. The default for `OEOverlap` is `Exact`. Alternatives are defined in the `OEOverlapMethod` namespace. (CROSSREF here)

SetRadiiApproximation

```
bool SetRadiiApproximation(unsigned int type)
```

Set the radius approximation used to calculate overlap. The default for `OEOverlap` is `Carbon`. Alternatives are defined in the `OEOverlapRadii` namespace. (CROSSREF here)

SetRefGrid

```
bool SetRefGrid(const OESystem::OEScalarGrid &refgrid)
```

Set a reference grid for the calculation. An internal copy is made. Any previous reference molecule or grid is cleared.

SetRefMol

```
bool SetRefMol(const OEChem::OEMolBase &refmol)
```

Set a reference molecule for the calculation. An internal copy is made. Any previous reference molecule or grid is cleared.

SetRepresentationLevel

```
bool SetRepresentationLevel(unsigned int type)
```

Set the representation level for the gaussians in `OEOverlap`. The default is `Atomic`. Alternatives are defined in the `OEOverlapRepresentation` namespace (CROSSREF here).

SetUseHydrogens

```
bool SetUseHydrogens(bool state)
```

Boolean to determine whether hydrogens are included in the shape calculation. By default this is false and hydrogens are ignored.

4.1.21 OEOverlapResults

```
struct OEOverlapResults
```

This class represents *OEOverlapResults*.

Constructors

```
OEOverlapResults()
```

Default and copy constructors.

GetFitTversky

```
float GetFitTversky() const
```

GetRefTversky

```
float GetRefTversky() const
```

GetTversky

```
float GetTversky(float alpha=0.95f, float beta=0.05f) const
```

4.2 OEShape Constants

4.2.1 OEBOMinType

This namespace contains constants.

Tanimoto

Overlap

Autoscale

Default

4.2.2 OEBOOrientation

This namespace contains constants.

Inertial

AsIs

Random

InertialAtHeavyAtoms

InertialAtColorAtoms

UserInertialStarts

Default

4.2.3 OEColorFFType

This namespace contains constants.

Undefined

OEDefault

ImplicitMillsDean

ExplicitMillsDean

Custom

Max

4.2.4 OEOverlapMethod

This namespace contains constants.

Grid

Analytic

Analytic2

Exact

Max

4.2.5 OEOverlapRadii

This namespace contains constants.

Carbon

Dual

All

Default

Max

4.2.6 OEOverlapRepresentation

This namespace contains constants.

Atomic

Atoms

Bonding

Bonds

Full

All

Gaussian

Default

Max

4.3 OEShape Functions

4.3.1 OEAddColorAtom

```

OEChem::OEAtomBase *OEAddColorAtom(OEChem::OEMolBase &mol, float *coords,
                                     unsigned type, const std::string &typeName,
                                     const std::string &parentName=std::string())
OEChem::OEAtomBase *OEAddColorAtom(OEChem::OEMolBase &mol,
                                     const OEChem::OEAtomBase **parents,
                                     unsigned num, unsigned type,
                                     const std::string &typeName)
OEChem::OEAtomBase *OEAddColorAtom(OEChem::OEMCMolBase &mol,
                                     const OEChem::OEAtomBase **parents,
                                     unsigned num, unsigned type,
                                     const std::string &typeName)

```

These functions allow adding color atoms manually, as opposed to being assigned via SMARTS matches from a color force field. Note that the color atom will either be placed at the coordinates supplied in the first version, or at the center of mass of the *parents* atoms. *num* is the number of atoms in the parent array.

4.3.2 OEAddColorAtoms

```

unsigned int OEAddColorAtoms(OEChem::OEMolBase &mol,
                             const OEColorForceField &cff,
                             bool setCachedSelfColor=false)
unsigned int OEAddColorAtoms(OEChem::OEMCMolBase &mol,
                             const OEColorForceField &cff,
                             bool setCachedSelfColor=false)

```

Add color atoms to the passed in molecule. Note that this is not normally required as both `OEColorOverlap` and `OEBestOverlay` make internal copies of molecules and add color atoms automatically, but if you continually re-use the same molecule, this can be an optimization. Additionally, setting `setCachedSelfColor` to true will pre-calculate and cache the self color on the molecule. This cached value will then be used by `OEColorOverlap` and `OEBestOverlay`. The value can be accessed via `OEGetCachedSelfColor`.

4.3.3 OEAddCoordsToColorAtom

```
bool OEAddCoordsToColorAtom(OEChem::OEMolBase &mol,
                             OEChem::OEAtomBase *coloratom, float *coords)
```

Set the 3D coordinates of the given color atom.

4.3.4 OECalcShapeMultipoles

```
bool OECalcShapeMultipoles(const OEChem::OEMolBase &mol, float *smult,
                            unsigned int type=OEOverlapRepresentation::Atoms,
                            bool useHydrogens=false)
```

Calculate the shape (steric) multipoles.

4.3.5 OECalcVolume

```
float OECalcVolume(const OEChem::OEMolBase &mol, bool useHydrogens=true)
```

Calculate the shape volume.

4.3.6 OEClearCachedSelfColor

```
void OEClearCachedSelfColor(OEChem::OEMCMolBase &mol)
void OEClearCachedSelfColor(OEChem::OEMolBase &mol)
```

Clear cached self color from molecule. See `OESetCachedSelfColor` for more info.

4.3.7 OEClearCachedSelfShape

```
void OEClearCachedSelfShape(OEChem::OEMCMolBase &mol)
void OEClearCachedSelfShape(OEChem::OEMolBase &mol)
```

Clear cached self shape from molecule. See `OESetCachedSelfShape` for more info.

4.3.8 OECompressColorAtoms

```
unsigned OECompressColorAtoms(OEChem::OEMolBase &mol)
```

Convert the color atoms on a molecule to an internal compressed form. Useful for saving space when storing pre-colored molecules. Use `OEUncompressColorAtoms` to restore the actual color atoms.

4.3.9 OECountColorAtoms

```
unsigned int OECountColorAtoms(const OEChem::OEMolBase &mol)
```

Return the number of color atoms attached to a molecule.

4.3.10 OEGetCachedSelfColor

```
float OEGetCachedSelfColor(const OEChem::OEConfBase &mol,
                           bool allcolor=false)
float OEGetCachedSelfColor(const OEChem::OEMolBase &mol,
                           bool allcolor=false)
```

Return the value of cached self color on the molecule or conformer. This function doesn't work for OEMCMolBase directly, so call it on each individual conformer. Returns 0.0 if there is no cached value. Use `OEHasCachedSelfColor` to determine if a real value is attached.

4.3.11 OEGetCachedSelfShape

```
float OEGetCachedSelfShape(const OEChem::OEConfBase &conf,
                           unsigned int radiiApprox = OEOverlapRadii::Carbon,
                           float cradius = 1.7f)
float OEGetCachedSelfShape(const OEChem::OEMolBase &mol,
                           unsigned int radiiApprox = OEOverlapRadii::Carbon,
                           float cradius = 1.7f)
```

Return the value of cached self shape on the molecule or conformer. The value for radii approximation and cradius need to match those used to set this value initially. This function doesn't work for OEMCMolBase directly, so call it on each individual conformer. Returns 0.0 if there is no cached value. Use `OEHasCachedSelfShape` to determine if a real value is attached.

4.3.12 OEGetColorAtoms

```
OESystem::OEIterBase<OEChem::OEAtomBase> *
  OEGetColorAtoms(OEChem::OEMolBase &mol)
OESystem::OEIterBase<const OEChem::OEAtomBase> *
  OEGetColorAtoms(const OEChem::OEMolBase &mol)
```

Return an iterator of all the color atoms attached to a molecule.

4.3.13 OEGetColorParents

```
OESystem::OEIterBase<OEChem::OEAtomBase> *
  OEGetColorParents(OEChem::OEAtomBase *atom)
OESystem::OEIterBase<const OEChem::OEAtomBase> *
  OEGetColorParents(const OEChem::OEAtomBase *atom)
```

Color atoms set via a SMARTS match store the matching atoms as *parents*. This method returns the parent atoms of a given color atom.

4.3.14 OEGetColorType

```
unsigned int OEGetColorType(const OEChem::OEAtomBase *coloratom)
```

Returns the type of a color atom as an unsigned int.

4.3.15 OEHasCachedSelfColor

```
bool OEHasCachedSelfColor(const OEChem::OEMCMolBase &mol)
```

```
bool OEHasCachedSelfColor(const OEChem::OEMolBase &mol)
```

Returns true if the molecule has cached self color.

4.3.16 OEHasCachedSelfShape

```
float OEHasCachedSelfShape(const OEChem::OEConfBase &conf,  
                           unsigned int radiiApprox = OEOverlapRadii::Carbon,  
                           float cradius = 1.7f)
```

```
float OEHasCachedSelfShape(const OEChem::OEMolBase &mol,  
                           unsigned int radiiApprox = OEOverlapRadii::Carbon,  
                           float cradius = 1.7f)
```

Returns true if the molecule has cached self shape. The value for radii approximation and cradius need to match those used to set this value initially.

4.3.17 OEHasColorAtoms

```
bool OEHasColorAtoms(const OEChem::OEMolBase &mol)
```

Returns true if the molecule has any color atoms attached.

4.3.18 OEIsColorAtom

```
bool OEIsColorAtom(const OEChem::OEAtomBase *atom)
```

Returns true if the given atom is a color atom.

4.3.19 OERemoveColorAtoms

```
void OERemoveColorAtoms(OEChem::OEMolBase &mol)
```

Remove all color atoms from the molecule.

4.3.20 OESelfColor

```
float OESelfColor(const OEChem::OEMolBase &mol, const OECOLORForceField &cff,
                 bool allcolor=false)
```

Calculate the self color score for the given molecule, using the passed in color force field.

4.3.21 OESetCachedSelfColor

```
bool OESetCachedSelfColor(OEChem::OEMCMolBase &mol,
                          const OECOLORForceField &cff)
bool OESetCachedSelfColor(OEChem::OEMolBase &mol,
                          const OECOLORForceField &cff)
```

Using the passed-in OECOLORForceField, calculate and store the self color on each molecule. For the OEMCMolBase, each conformer gets its own stored value. To retrieve the value, use `OEGetCachedSelfColor`. OECOLOROverlap and OEBestOverlay will use the cached value instead of re-calculating the self terms.

4.3.22 OESetCachedSelfShape

```
bool OESetCachedSelfShape(OEChem::OEMCMolBase &mcmol,
                          unsigned int radiiApprox = OEOVERLAPRadii::Carbon,
                          float cradius = 1.7f)
bool OESetCachedSelfShape(OEChem::OEMolBase &mol,
                          unsigned int radiiApprox = OEOVERLAPRadii::Carbon,
                          float cradius = 1.7f)
```

Store the self shape term on the molecule. For the OEMCMolBase, each conformer stores its own value. To retrieve the value, use `OEGetCachedSelfShape`. OEOVERLAP and OEBestOverlay will use the cached value instead of re-calculating the self terms.

4.3.23 OESetColorParents

```
bool OESetColorParents(OEChem::OEAtomBase *atom,
                      const OEChem::OEAtomBase **parents, unsigned num)
```

Set the parent atoms of a given color atom. Not normally used as parents are set for color atoms added via a SMARTS match in the color force field. But if you are manually creating color atoms, this is a way to set its parents.

4.3.24 OESetColorType

```
bool OESetColorType(OEChem::OEAtomBase *coloratom, unsigned type,
                   const std::string &typeName)
```

Set/changes the type of a given color atom.

4.3.25 OESetCoordsFromColorParents

```
bool OESetCoordsFromColorParents(OEChem::OEMCMolBase &mol)
bool OESetCoordsFromColorParents(OEChem::OEMolBase &mol)
```

Generates 3D coordinates for the given color atom based on the center of mass of that color atom's parents.

4.3.26 OEShapeGetArch

```
const char *OEShapeGetArch()
```

Returns the architecture of the current version of the Shape toolkit. Examples include:

- microsoft-win32-x86
- redhat-RHEL5-x86_64
- suse-SLES10-ppc32

4.3.27 OEShapeGetLicensee

```
bool OEShapeGetLicensee(std::string &licensee)
```

Returns the LICENSEE from the current valid Shape TK license.

4.3.28 OEShapeGetPlatform

```
const char *OEShapeGetPlatform()
```

Returns the platform, including compiler version, of the current version of the Shape toolkit. Examples include:

- microsoft-win32-msvc9-MD-x86
- redhat-RHEL5-g++4.1-x86_64
- suse-SLES10-g++4.3-ppc32

4.3.29 OEShapeGetRelease

```
const char *OEShapeGetRelease()
```

Returns the version of this toolkit release. For example: 1.6.1.

4.3.30 OEShapeGetSite

```
bool OEShapeGetSite(std::string &site)
```

Returns the SITE from the current valid Shape TK license.

4.3.31 OEShapeGetVersion

```
unsigned int OEShapeGetVersion()
```

Returns the build date of the toolkit as an unsigned int. For example: 20090227.

4.3.32 OEShapeIsLicensed

```
bool OEShapeIsLicensed(const char *feature=0, unsigned int *expdate=0)
```

Determine whether a valid license file is present. This function may be called without a legitimate run-time license to determine whether it is safe to call any of OEShape's functionality.

The `features` argument can be used to check for a valid license to OEShape along with that feature. For example, to verify that OEShape can be used from Python:

```
if (!OEShapeIsLicensed("python"))
    OThrow.Warning("OEShape is not licensed for the python feature");
```

The second argument can be used to get the expiration date of the license. This is an array of size three with the date returned as [day, month, year]. Even if the function returns false due to an expired license, the `expdate` will show that expiration date. A value of a zeroes implies that no license or date was found.

```
unsigned int expdate[3];
if (OEShapeIsLicensed(0, expdate))
{
    OThrow.Info("License expires: day: %d month: %d year: %d",
               expdate[0], expdate[1], expdate[2]);
}
```

4.3.33 OESortOverlayScores

```
void OESortOverlayScores(OESystem::OIter<OEBestOverlayScore> &dst,
                        OESystem::OIter<OEBestOverlayResults> &scores,
                        const OESystem::OEBinaryPredicate<OEBestOverlayScore,
                        OEBestOverlayScore> &sorter, int nbest=1,
                        bool conforder=false)
```

Sort the scores from multiple sets of `OEBestOverlayResults` into a single iterator of `OEBestOverlayScore`. The sorting function can be one of the pre-defined functors or a user-defined version.

The `nbest` default value of **1** implies that on the best overlay for each ref-fit conformer pair is returned. A value greater than one will mean multiple overlays for each ref-fit conformer pair will be kept.

Setting `conforder` to **true** forces the results to come out in conformer index order. If `nbest` is greater than **1** and `conforder` is **true**, then for each ref-fit conformer pair, the scores will be sorted by the sorting function, but each set of ref-fit results will come out together.

Several of the `OEBestOverlay` examples show variations of these values. Perhaps the easiest way to understand them is to modify one of the examples and observe the change in the number and order of the scores.

4.3.34 OEUncompressColorAtoms

```
unsigned OEUncompressColorAtoms(OEChem::OEMolBase &mol)
```

Convert the internal compressed color atoms to actual color atoms. Has no effect unless the color atoms were previously compressed by `OECompressColorAtoms`

Note: This function was formerly named `OEStringToColorAtoms`.

RELEASE NOTES

5.1 ShapeTK 1.8.1

5.1.1 Bug fixes

- Fixing a crash in `OECalcVolume` when dealing with Zinc and dummy atoms.
- `OECalcVolume` will now default to using appropriate Bondii VdW radii when no radii are set on the molecule. This effectively changes the value returned this function most molecules.

5.2 ShapeTK 1.8.0

5.2.1 New features

- Added new functions that will cache the self shape score on a molecule. Analogous to the caching of self color, introduced in 1.7.2, this can speed up repeat shape calculations on the same molecule by not recalculating the self terms.
 - `OESetCachedSelfShape`
 - `OEHasCachedSelfShape`
 - `OEGetCachedSelfShape`
 - `OEClearCachedSelfShape`
- Updated the functions that cache self color to store work with multi-conformer molecules, storing a self color for each conformer.
- Added two new color force fields, `ImplicitMillsDeanNoRings` and `ExplicitMillsDeanNoRings`. These are identical to the existing similarly named force fields, but with no “rings” atom types or interactions.
- Added `Has` and `Delete` functions to the API for dealing with compressed color atoms.
- Several of the examples have been updated to count the output better and to use a better sort functor when color is included.
- Add new methods to `OEBestOverlayScore` to allow access to the transform.
 - `OEBestOverlayScore::GetRotMatrix`
 - `OEBestOverlayScore::GetTranslation`

5.2.2 Bug fixes

- Added `OEGetColorAtoms` and `OEGetColorParents` to Python, Java and C#.
- Fixed a bug where non-deterministic results could sometime occur when using `OEBestOverlay` and a highly symmetric molecule.
- Fixed calculation of multipoles. Previous versions used a non-irreducible representation of multipoles that is non-standard. This method now calculates standard multipoles and allow better comparison between the values of two molecules.
- Fixed a missing type in `ImplicitMillsDean` to match an aldehyde acceptor with or without the explicit proton.
- Fixed a bug where `OECalcVolume` could crash using a molecule with deleted atoms.

5.3 ShapeTK 1.7.2

5.3.1 New features

- `OEAddColorAtoms` now has an optional third argument to cause precalculation and caching of the self color. When re-using molecules in multiple comparisons (for example NxN studies), use `OEAddColorAtoms` once and then `OEBestOverlay` and `OECOLOROverlap` will use the pre-stored color atoms and self color to speed up the calculations.
- Added a set of functions to manipulate cached self color.
 - `OESetCachedSelfColor`
 - `OEHasCachedSelfColor`
 - `OEGetCachedSelfColor`
 - `OEClearCachedSelfColor`
- The pair of functions (`OECOLORAtomsToString` and `OESTringToColorAtoms`) added in v1.7.1 have been renamed to more appropriate names. The new functions are `OECompressColorAtoms` and `OEUncompressColorAtoms`.

5.3.2 Bug fixes

- Fixed a bug where dummy atoms could cause an erroneous result for `OECalcVolume`.

5.4 ShapeTK 1.7.1

5.4.1 New features

- Added new types in `OEBOOrientation`. All of these are designed to provide a more deterministic search over the reference molecule for those case where the size of the fit molecule is much smaller than the reference, for example, when trying to match a fragment into part of a reference molecule.
 - `OEBOOrientation::InertialAtHeavyAtoms` moves the center of mass of the fit molecule to each reference molecule heavy atom and performs 4 inertial starts at the position. This results in many more starting positions, but provides a more direct way to search over an entire reference molecule, without resorting to random starts.

- `OEBOrientation::InertialAtColorAtoms` performs a similar search as above, but just moves to the location of each reference molecule color atom.
- `OEBOrientation::UserInertialStarts`, used in conjunction with `OEBestOverlay::SetUserStarts` allows the user to pick specific points in space to perform the 4 inertial starts.
- Fixed a bug when calculating Tanimoto while using a grid as the reference object.
- Added more functions to manipulate the color atoms on a molecule. These include the ability to add color atoms one at a time (`OEAddColorAtom`) and the ability to get an iterator of color atoms from a molecule (`OEGetColorAtoms`).
- Added a pair of functions (`OECColorAtomsToString` and `OEStrngToColorAtoms`) that allow converting the color atoms of a molecule into a compressed string representation (that is attached to the molecule) and the to restore the actual color atoms from that string.

5.4.2 Bug fixes

- Fixed a bug that could cause a crash when passing an empty molecule into `OECalcVolume` or `OECalcShapeMultipoles`.
- Fixed a bug that could cause a crash when passing large molecules to `OECalcVolume`.

5.5 ShapeTK 1.7.0

5.5.1 New features

- Color Tanimoto is calculated just like Shape Tanimoto by using the self color scores of each molecule combined with the actual color score. The end result is to better match molecules of the same color, penalizing a molecule for having too much as well as too little color. Color Tversky scores can be calculated as well. We now calculate these extra color scores in `OECColorOverlap` and `OEBestOverlay`.
- `OECColorResults` now has Tanimoto, Tversky, RefTversky and FitTversky.
- `OEBestOverlayScore` now has ColorTanimoto, ColorTversky, RefColorTversky, FitColorTversky and adds three new combo scores based on these:
 - TanimotoCombo - the sum of Shape Tanimoto and Color Tanimoto
 - RefTverskyCombo - the sum of Ref Shape Tversky and Ref Color Tversky
 - FitTverskyCombo - the sum of Fit Shape Tversky and Fit Color Tversky
- `OEOverlapResults` now has Tversky, RefTversky and FitTversky.

5.5.2 Bug fixes

- Fixed an internal memory leak in `OECColorOverlap/OEBestOverlay` color objects.
- Fixed a bug where in some cases the user-supplied value for carbon radius in `OEOverlap` was ignored.

5.6 ShapeTK 1.6.2

5.6.1 New features

- Added the ability to get the expiration date from `OEShapeIsLicensed` function.

5.6.2 Bug fixes

- Fixed a memory leak in the calculation of self color in `OEColorFunc`.
- Fixed a bug that prevented pre-colored molecules from being used in `OEColorFunc`.
- Replace old error messages that just went to `stderr` and routed all errors and warnings via `OEThrow`.

5.7 ShapeTK 1.6.1

This is the first release as part of the consolidated toolkits release.

5.7.1 New features

- Added functions to retrieve the version and build date of the library.

5.7.2 Bug fixes

- Changed to a stable sort inside `OESortOverlayScores` to prevent crashes with a very large number of results.
- Fixes to `OEOverlap` that in some cases could result in slow behavior when internal grids were rebuilt instead of being cached.
- Fix a subtle bug that could result in erroneous behavior if reusing the same `OEOverlap` object while changing the fit object from a grid to a molecule or from a molecule to a grid.

5.8 ShapeTK 1.6

5.8.1 New features

- Added new functions to calculate steric volume (`OECalcVolume`) and steric multipoles (`OECalcShapeMultipoles`).

5.8.2 Bug fixes

- Fixed a bug where `OEAddColorAtoms` would not correctly add color atoms to a molecule that had the `IntType` field on atoms filled from a previous calculation. `IntType` is scratch space on atoms, so `OEAddColorAtoms` now clears the `IntType` before adding color atoms to a molecule.
- Changed one of the cation definitions in `ImplicitMillsDean` force field from the too general (“n:lanaa1”) to the more specific (“n:lcnc1”).

- Fixed a bug that caused `OEBOrientation::AsIs` to not always start from the passed in positions.

5.9 ShapeTK 1.5

5.9.1 New features

- First release in C++, Java and Python.

5.10 Indices and tables

- *Index*
- *Search Page*

BIBLIOGRAPHY

- [Grant-1995] J.A. Grant and B.T. Pickup, **A Gaussian Description of Molecular Shape**, *Journal of Physical Chemistry*, Vol. 99, pp. 3503-3510, **1995**.
- [Grant-1996] J.A. Grant, M.A. Gallardo and B.T. Pickup, **A fast method of molecular shape comparison. A simple application of a Gaussian description of molecular shape**, *Computational Chemistry*, Vol. 17, pp. 1653–1666, **1996**.
- [Grant-1997] J.A. Grant and B.T. Pickup, **Gaussian Shape Methods**, *Computer Simulation of Biomolecular Systems*, Vol 3, **1997**.
- [MillsDean-1996] J.E.J. Mills and P.M. Dean, **Three-dimensional hydrogen-bond geometry and probability information from a crystal survey**, *Journal of Computer-Aided Molecular Design*, Vol. 10, pp. 607, **1996**.

INDEX

A

AddColorer
 OEShape::OECOLORForceField, 37
AddInteraction
 OEShape::OECOLORForceField, 37
AddScore
 OEShape::OEBestOverlayResults, 34
AddType
 OEShape::OECOLORForceField, 37

B

bestoverlay-color.cpp
 Example Code, 20
bestoverlay1.cpp
 Example Code, 15
bestoverlay2.cpp
 Example Code, 17
bestoverlay3.cpp
 Example Code, 18
bestoverlay4.cpp
 Example Code, 19

C

Clear
 OEShape::OEBestOverlayResults, 34
 OEShape::OECOLORForceField, 37
ClearColorForceField
 OEShape::OEBestOverlay, 28
ClearInteractions
 OEShape::OECOLORForceField, 37
ClearUserStarts
 OEShape::OEBestOverlay, 28
coloroverlap.cpp
 Example Code, 10
ColorScore
 OEShape::OECOLOROverlap, 38
colorscore.cpp
 Example Code, 11
Constructors
 OEShape::OEBestOverlay, 28
 OEShape::OEBestOverlayResults, 34

OEShape::OEBestOverlayScore, 34
OEShape::OECOLORForceField, 36
OEShape::OECOLOROverlap, 38
OEShape::OECOLORResults, 39
OEShape::OEIsColorAtomPred, 44
OEShape::OEOverlap, 45
OEShape::OEOverlapResults, 48

CreateCopy

OEShape::OEHighestColorTanimoto, 40
OEShape::OEHighestComboScore, 40
OEShape::OEHighestFitColorTversky, 41
OEShape::OEHighestFitTversky, 41
OEShape::OEHighestFitTverskyCombo, 42
OEShape::OEHighestOverlap, 42
OEShape::OEHighestRefColorTversky, 42
OEShape::OEHighestRefTversky, 43
OEShape::OEHighestRefTverskyCombo, 43
OEShape::OEHighestScaledColor, 43
OEShape::OEHighestTanimoto, 44
OEShape::OEHighestTanimotoCombo, 44
OEShape::OEIsColorAtomPred, 45

E

Example Code

bestoverlay-color.cpp, 20
bestoverlay1.cpp, 15
bestoverlay2.cpp, 17
bestoverlay3.cpp, 18
bestoverlay4.cpp, 19
coloroverlap.cpp, 10
colorscore.cpp, 11
nxnshape.cpp, 22
rescore.cpp, 8
shapeprops.cpp, 24
simpleoverlap.cpp, 7
usercolor.cpp, 12

G

GetAllColor

OEShape::OEBestOverlay, 28
OEShape::OECOLOROverlap, 38

- GetCarbonRadius
 - OEShape::OEBestOverlay, 28
 - OEShape::OEOverlap, 45
 - GetColorTanimoto
 - OEShape::OEBestOverlayScore, 35
 - GetColorTversky
 - OEShape::OEBestOverlayScore, 35
 - GetComboScore
 - OEShape::OEBestOverlayScore, 35
 - GetFitColorTversky
 - OEShape::OEBestOverlayScore, 35
 - GetFitTversky
 - OEShape::OEBestOverlayScore, 35
 - OEShape::OECOLORResults, 39
 - OEShape::OEOverlapResults, 48
 - GetFitTverskyCombo
 - OEShape::OEBestOverlayScore, 35
 - GetHandle
 - OEShape::OEOverlap, 45
 - GetInertialAxialDivisions
 - OEShape::OEBestOverlay, 29
 - GetInitialOrientation
 - OEShape::OEBestOverlay, 29
 - GetMaxRandomTranslation
 - OEShape::OEBestOverlay, 29
 - GetMethod
 - OEShape::OEBestOverlay, 29
 - OEShape::OEOverlap, 46
 - GetMinimizeType
 - OEShape::OEBestOverlay, 29
 - GetNumRandomStarts
 - OEShape::OEBestOverlay, 29
 - GetNumUserStarts
 - OEShape::OEBestOverlay, 29
 - GetRadiiApproximation
 - OEShape::OEBestOverlay, 29
 - OEShape::OEOverlap, 46
 - GetRandomSeed
 - OEShape::OEBestOverlay, 30
 - GetRefColorTversky
 - OEShape::OEBestOverlayScore, 35
 - GetRefGrid
 - OEShape::OEBestOverlay, 30
 - GetRefMol
 - OEShape::OEBestOverlay, 30
 - GetRefSelfColor
 - OEShape::OEBestOverlay, 30
 - GetRefSymmetry
 - OEShape::OEBestOverlay, 30
 - GetRefTversky
 - OEShape::OEBestOverlayScore, 35
 - OEShape::OECOLORResults, 39
 - OEShape::OEOverlapResults, 48
 - GetRefTverskyCombo
 - OEShape::OEBestOverlayScore, 35
 - GetRepresentationLevel
 - OEShape::OEBestOverlay, 30
 - OEShape::OEOverlap, 46
 - GetRotMatrix
 - OEShape::OEBestOverlayScore, 36
 - GetScores
 - OEShape::OEBestOverlayResults, 34
 - GetSelfColor
 - OEShape::OECOLOROverlap, 38
 - GetSymmetryThreshold
 - OEShape::OEBestOverlay, 30
 - GetTanimoto
 - OEShape::OECOLORResults, 40
 - GetTanimotoCombo
 - OEShape::OEBestOverlayScore, 36
 - GetTitle
 - OEShape::OECOLORForceField, 37
 - GetTranslation
 - OEShape::OEBestOverlayScore, 36
 - GetTversky
 - OEShape::OEBestOverlayScore, 36
 - OEShape::OECOLORResults, 40
 - OEShape::OEOverlapResults, 48
 - GetType
 - OEShape::OECOLORForceField, 37
 - GetTypeName
 - OEShape::OECOLORForceField, 37
 - GetUseHydrogens
 - OEShape::OEBestOverlay, 30
 - OEShape::OEOverlap, 46
 - GetUserStarts
 - OEShape::OEBestOverlay, 31
- I**
- Init
 - OEShape::OECOLORForceField, 37
- N**
- nxnshape.cpp
 - Example Code, 22
- O**
- OEShape::OEAddColorAtom, 51
 - OEShape::OEAddColorAtoms, 51
 - OEShape::OEAddCoordsToColorAtom, 52
 - OEShape::OEBestOverlay, 27
 - ClearColorForceField, 28
 - ClearUserStarts, 28
 - Constructors, 28
 - GetAllColor, 28
 - GetCarbonRadius, 28
 - GetInertialAxialDivisions, 29
 - GetInitialOrientation, 29

- GetMaxRandomTranslation, 29
- GetMethod, 29
- GetMinimizeType, 29
- GetNumRandomStarts, 29
- GetNumUserStarts, 29
- GetRadiiApproximation, 29
- GetRandomSeed, 30
- GetRefGrid, 30
- GetRefMol, 30
- GetRefSelfColor, 30
- GetRefSymmetry, 30
- GetRepresentationLevel, 30
- GetSymmetryThreshold, 30
- GetUseHydrogens, 30
- GetUserStarts, 31
- operator=, 28
- Overlay, 31
- SetAllColor, 31
- SetCarbonRadius, 31
- SetColorForceField, 31
- SetColorOptimize, 31
- SetInertialAxialDivisions, 31
- SetInitialOrientation, 32
- SetMaxRandomTranslation, 32
- SetMethod, 32
- SetMinimizeType, 32
- SetNumRandomStarts, 32
- SetRadiiApproximation, 32
- SetRandomSeed, 32
- SetRefGrid, 33
- SetRefMol, 33
- SetRepresentationLevel, 33
- SetSymmetryThreshold, 33
- SetUseHydrogens, 33
- SetUserStarts, 33
- OEShape::OEBestOverlayResults, 33
 - AddScore, 34
 - Clear, 34
 - Constructors, 34
 - GetScores, 34
 - operator=, 34
- OEShape::OEBestOverlayScore, 34
 - Constructors, 34
 - GetColorTanimoto, 35
 - GetColorTversky, 35
 - GetComboScore, 35
 - GetFitColorTversky, 35
 - GetFitTversky, 35
 - GetFitTverskyCombo, 35
 - GetRefColorTversky, 35
 - GetRefTversky, 35
 - GetRefTverskyCombo, 35
 - GetRotMatrix, 36
 - GetTanimotoCombo, 36
- GetTranslation, 36
- GetTversky, 36
 - operator=, 34
 - Transform, 36
- OEShape::OEBOMinType, 48
 - OEShape::OEBOMinType::Autoscale, 48
 - OEShape::OEBOMinType::Default, 48
 - OEShape::OEBOMinType::Overlap, 48
 - OEShape::OEBOMinType::Tanimoto, 48
- OEShape::OEBOOrientation, 48
 - OEShape::OEBOOrientation::AsIs, 49
 - OEShape::OEBOOrientation::Default, 49
 - OEShape::OEBOOrientation::Inertial, 49
 - OEShape::OEBOOrientation::InertialAtColorAtoms, 49
 - OEShape::OEBOOrientation::InertialAtHeavyAtoms, 49
 - OEShape::OEBOOrientation::Random, 49
 - OEShape::OEBOOrientation::UserInertialStarts, 49
- OEShape::OECalcShapeMultipoles, 52
- OEShape::OECalcVolume, 52
- OEShape::OEClearCachedSelfColor, 52
- OEShape::OEClearCachedSelfShape, 52
- OEShape::OECColorFFType, 49
 - OEShape::OECColorFFType::Custom, 49
 - OEShape::OECColorFFType::ExplicitMillsDean, 49
 - OEShape::OECColorFFType::ImplicitMillsDean, 49
 - OEShape::OECColorFFType::Max, 49
 - OEShape::OECColorFFType::OEDefault, 49
 - OEShape::OECColorFFType::Undefined, 49
- OEShape::OECColorForceField, 36
 - AddColorer, 37
 - AddInteraction, 37
 - AddType, 37
 - Clear, 37
 - ClearInteractions, 37
 - Constructors, 36
 - GetTitle, 37
 - GetType, 37
 - GetTypeName, 37
 - Init, 37
 - operator=, 36
 - Ready, 38
 - SetTitle, 38
 - Write, 38
- OEShape::OECColorOverlap, 38
 - ColorScore, 38
 - Constructors, 38
 - GetAllColor, 38
 - GetSelfColor, 38
 - operator=, 38
 - SetAllColor, 39
 - SetColorForceField, 39
 - SetFitMol, 39
 - SetRefMol, 39
- OEShape::OECColorResults, 39

- Constructors, 39
- GetFitTversky, 39
- GetRefTversky, 39
- GetTanimoto, 40
- GetTversky, 40
- OEShape::OECompressColorAtoms, 52
- OEShape::OECountColorAtoms, 53
- OEShape::OEGetCachedSelfColor, 53
- OEShape::OEGetCachedSelfShape, 53
- OEShape::OEGetColorAtoms, 53
- OEShape::OEGetColorParents, 53
- OEShape::OEGetColorType, 54
- OEShape::OEHasCachedSelfColor, 54
- OEShape::OEHasCachedSelfShape, 54
- OEShape::OEHasColorAtoms, 54
- OEShape::OEHighestColorTanimoto, 40
 - CreateCopy, 40
 - operator(), 40
- OEShape::OEHighestComboScore, 40
 - CreateCopy, 40
 - operator(), 40
- OEShape::OEHighestFitColorTversky, 41
 - CreateCopy, 41
 - operator(), 41
- OEShape::OEHighestFitTversky, 41
 - CreateCopy, 41
 - operator(), 41
- OEShape::OEHighestFitTverskyCombo, 41
 - CreateCopy, 42
 - operator(), 41
- OEShape::OEHighestOverlap, 42
 - CreateCopy, 42
 - operator(), 42
- OEShape::OEHighestRefColorTversky, 42
 - CreateCopy, 42
 - operator(), 42
- OEShape::OEHighestRefTversky, 42
 - CreateCopy, 43
 - operator(), 43
- OEShape::OEHighestRefTverskyCombo, 43
 - CreateCopy, 43
 - operator(), 43
- OEShape::OEHighestScaledColor, 43
 - CreateCopy, 43
 - operator(), 43
- OEShape::OEHighestTanimoto, 44
 - CreateCopy, 44
 - operator(), 44
- OEShape::OEHighestTanimotoCombo, 44
 - CreateCopy, 44
 - operator(), 44
- OEShape::OEIsColorAtom, 54
- OEShape::OEIsColorAtomPred, 44
 - Constructors, 44
 - CreateCopy, 45
 - operator(), 45
- OEShape::OEOverlap, 45
 - Constructors, 45
 - GetCarbonRadius, 45
 - GetHandle, 45
 - GetMethod, 46
 - GetRadiiApproximation, 46
 - GetRepresentationLevel, 46
 - GetUseHydrogens, 46
 - operator=, 45
 - Overlap, 46
 - SetCarbonRadius, 46
 - SetFitGrid, 46
 - SetFitMol, 47
 - SetMethod, 47
 - SetRadiiApproximation, 47
 - SetRefGrid, 47
 - SetRefMol, 47
 - SetRepresentationLevel, 47
 - SetUseHydrogens, 47
- OEShape::OEOverlapMethod, 49
- OEShape::OEOverlapMethod::Analytic, 50
- OEShape::OEOverlapMethod::Analytic2, 50
- OEShape::OEOverlapMethod::Exact, 50
- OEShape::OEOverlapMethod::Grid, 50
- OEShape::OEOverlapMethod::Max, 50
- OEShape::OEOverlapRadii, 50
- OEShape::OEOverlapRadii::All, 50
- OEShape::OEOverlapRadii::Carbon, 50
- OEShape::OEOverlapRadii::Default, 50
- OEShape::OEOverlapRadii::Dual, 50
- OEShape::OEOverlapRadii::Max, 50
- OEShape::OEOverlapRepresentation, 50
- OEShape::OEOverlapRepresentation::All, 51
- OEShape::OEOverlapRepresentation::Atomic, 51
- OEShape::OEOverlapRepresentation::Atoms, 51
- OEShape::OEOverlapRepresentation::Bonding, 51
- OEShape::OEOverlapRepresentation::Bonds, 51
- OEShape::OEOverlapRepresentation::Default, 51
- OEShape::OEOverlapRepresentation::Full, 51
- OEShape::OEOverlapRepresentation::Gaussian, 51
- OEShape::OEOverlapRepresentation::Max, 51
- OEShape::OEOverlapResults, 48
 - Constructors, 48
 - GetFitTversky, 48
 - GetRefTversky, 48
 - GetTversky, 48
- OEShape::OERemoveColorAtoms, 54
- OEShape::OESelfColor, 54
- OEShape::OESetCachedSelfColor, 55
- OEShape::OESetCachedSelfShape, 55
- OEShape::OESetColorParents, 55
- OEShape::OESetColorType, 55

- OEShape::OESetCoordsFromColorParents, 55
- OEShape::OEShapeGetArch, 56
- OEShape::OEShapeGetLicensee, 56
- OEShape::OEShapeGetPlatform, 56
- OEShape::OEShapeGetRelease, 56
- OEShape::OEShapeGetSite, 56
- OEShape::OEShapeGetVersion, 56
- OEShape::OEShapeIsLicensed, 57
- OEShape::OESortOverlayScores, 57
- OEShape::OEUncompressColorAtoms, 57
- operator()
 - OEShape::OEHighestColorTanimoto, 40
 - OEShape::OEHighestComboScore, 40
 - OEShape::OEHighestFitColorTversky, 41
 - OEShape::OEHighestFitTversky, 41
 - OEShape::OEHighestFitTverskyCombo, 41
 - OEShape::OEHighestOverlap, 42
 - OEShape::OEHighestRefColorTversky, 42
 - OEShape::OEHighestRefTversky, 43
 - OEShape::OEHighestRefTverskyCombo, 43
 - OEShape::OEHighestScaledColor, 43
 - OEShape::OEHighestTanimoto, 44
 - OEShape::OEHighestTanimotoCombo, 44
 - OEShape::OEIsColorAtomPred, 45
- operator=
 - OEShape::OEBestOverlay, 28
 - OEShape::OEBestOverlayResults, 34
 - OEShape::OEBestOverlayScore, 34
 - OEShape::OECColorForceField, 36
 - OEShape::OECColorOverlap, 38
 - OEShape::OEOverlap, 45
- Overlap
 - OEShape::OEOverlap, 46
- Overlay
 - OEShape::OEBestOverlay, 31
- R**
- Ready
 - OEShape::OECColorForceField, 38
- rescore.cpp
 - Example Code, 8
- S**
- SetAllColor
 - OEShape::OEBestOverlay, 31
 - OEShape::OECColorOverlap, 39
- SetCarbonRadius
 - OEShape::OEBestOverlay, 31
 - OEShape::OEOverlap, 46
- SetColorForceField
 - OEShape::OEBestOverlay, 31
 - OEShape::OECColorOverlap, 39
- SetColorOptimize
 - OEShape::OEBestOverlay, 31
- SetFitGrid
 - OEShape::OEOverlap, 46
- SetFitMol
 - OEShape::OECColorOverlap, 39
 - OEShape::OEOverlap, 47
- SetInertialAxialDivisions
 - OEShape::OEBestOverlay, 31
- SetInitialOrientation
 - OEShape::OEBestOverlay, 32
- SetMaxRandomTranslation
 - OEShape::OEBestOverlay, 32
- SetMethod
 - OEShape::OEBestOverlay, 32
 - OEShape::OEOverlap, 47
- SetMinimizeType
 - OEShape::OEBestOverlay, 32
- SetNumRandomStarts
 - OEShape::OEBestOverlay, 32
- SetRadiiApproximation
 - OEShape::OEBestOverlay, 32
 - OEShape::OEOverlap, 47
- SetRandomSeed
 - OEShape::OEBestOverlay, 32
- SetRefGrid
 - OEShape::OEBestOverlay, 33
 - OEShape::OEOverlap, 47
- SetRefMol
 - OEShape::OEBestOverlay, 33
 - OEShape::OECColorOverlap, 39
 - OEShape::OEOverlap, 47
- SetRepresentationLevel
 - OEShape::OEBestOverlay, 33
 - OEShape::OEOverlap, 47
- SetSymmetryThreshold
 - OEShape::OEBestOverlay, 33
- SetTitle
 - OEShape::OECColorForceField, 38
- SetUseHydrogens
 - OEShape::OEBestOverlay, 33
 - OEShape::OEOverlap, 47
- SetUserStarts
 - OEShape::OEBestOverlay, 33
- shapeprops.cpp
 - Example Code, 24
- simpleoverlap.cpp
 - Example Code, 7
- T**
- Transform
 - OEShape::OEBestOverlayScore, 36
- U**
- usercolor.cpp
 - Example Code, 12

W

Write

 OEShape::OEColorForceField, 38