Shape Tester. What we need?

1) Tests:

a)Test individual shapes

Correctness and consistency(like original SBT test , OpticalEscape, ShapeChecker)

Accuracy(precision of methods, like for DistanceToIn() in SurfaceChecker)

b)Comparison between different packages (unified SBT)

Performance

Comparison of values

2) For testing we need to **generate points and directions and store them for performance test.**

3) Report results of the tests with possibility to store some values

b)Correctness , in case of error store point,direction,value of difference

c)Accuracy , report accuracy per method, some staistics

a)Time per package and differences per package

Should standardize the output of the tests in order to be able to visualize them with a single program.

4) In addition we need Unit Test with 'asserts' for each shape.

5)Automatisation of part of testing suite for regular use

Shape Tester. Generation of test rays.

What we use/need :

Points : Inside, created randomly by using existing Extent() and verify that points are really Inside by using Inside().

Outside, created randomly by using given dimensions of World ounds and verify that points are really Outside by using Inside().

Surface, GetPointOnSurface() already implemented for all USolids.

Edges, GetPointsOnEdges() will be implemented for all USolids.

Directions: Random, created randomly directions using angles phi and theta.

Random specifics , ensuring that ray will go Inside/Outside/Surface/Edge.

Random with reflection(or going back) for Optical Escape test.

We want to be able to use different proportions for different methods or tests.

Shape Tester.Generation of test rays.

CreatePointsInside(const int n, const double r, UVector3 &points);

CreatePointsOutside(const int n, const double r, UVector3 &points);
//r = dimensions of bounds

CreatePointsOnSurface(const int n, UVector3 &points);

CreatePointsOnEdges(const int n, UVector3 &points);

CreateRandomDirections(const int n, UVector3 &directions);

CreateSpecificDirections(const int n, const int state,
UVector3 &directions);
//State = Test Ray going Inside/Outside/Surface/Edge of shape (enum?)

Shape Tester. Individual methods.

bool TestMethod(**n,parametersPoints,parametersDirections, verbose**)
{
  points = CreatePoints(n,parametersPoints);
  directions = CreateDirections(n,parameterDirections); //if needed
  debug = InitializeDebugger();
  for (int i = 0; i < n; i++) TestOnePoint (i,points, directions, verbose, debug);
  ReportErrors (verbose, debug);
}

n = number of points to test

parametersPoints = percentage of Points generated Inside/Outside/Surface/Edge

parametersDirections = percentage of Directions pointing Inside/Outside/Surface/Edge

verbose = level of verbose:

0=just report minimum information(how many errors are found)

n=store and report results, n defines how many errors we want to store and report

debug = pointer to stored values of errors

for each error store :specification of the error, point,direction, value, value expected, number of stored points depend on verbose parameter

Shape Tester. Example of output.

```
% SBT logged output Thu Mar 13 11:42:39 2014
% /solid/G4Polycone2 0 360 18 (-6781,-6735,-6735,-6530,-6530,-4185,-4185,-3475,-
3475,3475,3475,4185,4185,6530,6530,6735,6735,6781)
(41,41,41,41,41,41,120,120,1148,1148,120,120,41,41,41,41,41,41)
(415,415,3275,3275,4251,4251,4251,4251,4251,4251,4251,4251,4251,4251,3275,3275,415,415)

% maxPoints = 10000
% maxErrors = 100
% End of test (maximum number points) Thu Mar 13 11:42:50 2014
% Statistics: points=10000 errors=0 errors reported=0
%           inside=5217 outside=4783 surface=0
%           cpu time=11
%(End of file)
```

Example of debug output:

Method InsidePoins() :points tested= 10000,errors found = 100, errors reported =1

Point1:'point not inside', point=(1,1,1), value=eSurface, expected=eInside

Shape Tester. Individual Shape Tests.

1)Tests of correctness of the answer of main methods for points Inside,Outside or on the Surface/Edge. This test can be used together with unit tests for automatized test of library.

testPointsInside(n,parametersPoints,parametersDirections, verbose)
testPointsOutside(n,parametersPoints,parametersDirections, verbose)
testPointsOnSurface(n,parametersPoints,parametersDirections, verbose)

2)Test of correctness and accuracy of one method.Test can give some statistics about precision of the methods, store values for histograms.
Will include tests for 'far away' and 'close by' points.

testDistanceToIn(n,parametersPoints,parametersDirections, verbose, dist)
testDistanceToOut(n,parametersPoints,parametersDirections, verbose,dist)
double dist = dist from which we will shoot rays
testSafety(n,parametersPoints,parametersDirections, verbose,nSphere, precise, useDistanceToIn)
int nSphere =how many points use for test“Safety Sphere”
bool precise = precise or not precise Safety
bool useDistanceToIn = compare or not with DistanceToIn
testNormal (n,parametersPoints, verbose,nReflections) test similar to Optical Escape
nReflections = number of Reflections per point
testVoxels(n,parametersPoints, verbose)
testConvexity(?, this test not exist yet, exist only comparison of the packages)

Shape Tester. Example.TestOutsidePoints().

- Consider pair of points, one inside and one outside the solid
- Select direction versus inside
- Must be DistanceToIn() < infinity (and less the distance between the two points)
- Compute SafetyFromOutside() (d1), cannot be bigger than DistanceToIn()
- Propagate to surface by d1, and compute Inside(); it must give 'kSurface', otherwise report overshoot or undershoot
- Compute SafetyFromIn(), must be zero
- Compute DistanceToOut(), should not be zero and be smaller than the extent along that direction
- Compute DistanceToIn(), has to be zero
- Invert direction (towards outside)
- Compute SafetyFromOutside(), must be zero
- Compute DistanceToIn(), must be greater than zero
- Compute DistanceToOut(), must be zero
- Compare the computed normals and make the dot product with direction, must be negative

Shape Tester. Example.TestSafety().

- Generate random points inside 2*BoundingBox
- Compute Safety() for each of them and generate random directions
- Move by Safety-tolerance and calculate Inside() for new point

 and for original point, calculated values of Inside have to be the same

- If demanded in test, for each point and direction,

compute the distance to the solid (DistanceToIn() or DistanceToOut()

depending on Inside() result)

- Compare the distance to the value of the safety; must be distance > safety
- Store found differences

Shape Tester. Example.TestNormal().

- Generate random points inside the solid and random directions for each of them
- For each propagate by DistToOut() to surface and reflect randomly

so that (norm).(dir)<0 and compute DistToOut()

- DistToOut() must never be zero
- Corners and edges will be most exercised

Shape Tester. Example.AccuracyTest().

- **Accuracy test for DistanceToIn/Out()**
- Generate random points in the world (may be inside or outside the solid)
- Generate random point on surface
- Calculate DistanceToIn() from any point outside in direction

to the point on surface, taking care of multiple intersections

- Compute numerical error comparing to the real distance, store results
- Same for points inside the solid for DistanceToOut()

Shape Tester. Comparison .

Comparison of performance and values with different packages.
Will be nice to have it also, may be in different place?

To have comparison directly in 'ShapeTester':

-We need to create corresponding shapes in different package
-We need to store points and directions for performance comparison:

```
bool TestMethod(n,parametersPoints,parametersDirections, verbose, shape, package)
{
  points = CreatePoints(n,parametersPoints);
  directions = CreateDirections(n,parameterDirections); //if needed
  SetupShape(shape,package);
  storing = InitializeStoring(); //storing differences
  TestMethod (n,points, directions, verbose, storing);
  ReportTime();
  ReportDifferences(verbose, storing);

}
```