CollDet Reference Manual

1.0

Generated by Doxygen 1.5.2

Tue Oct 16 18:12:21 2007

Contents

Chapter 1

Collision Detection

This is a Collision Detection library to use with OpenSG.To get some information about the algorithms that are used in the library have a look at the site of the author, Gabriel Zachmann.

The usage of this software is very simple:

1. include the file *Collision.h*;
2. create a class inherited from *col::Callback*. This class should have an *operator ()*, where you can implement everything that should happen when two of your objects have collided;
3. initialize the library by creating an instance of the class *col::CollisionPipeline*;
4. register your objects with the library using *col::CollisionPipeline::makeCollidable*.
   That's all.

For question or comments, send a mail to zach@tu-clausthal.de

Clausthal, 01.08.2007

Chapter 2

CollDet Namespace Index

2.1 CollDet Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

CollDet Hierarchical Index

3.1 CollDet Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

CollDet Class Index

4.1 CollDet Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

CollDet File Index

5.1 CollDet File List

Here is a list of all documented files with brief descriptions:

Chapter 6

CollDet Page Index

6.1 CollDet Related Pages

Here is a list of all related documentation pages:

Chapter 7

CollDet Namespace Documentation

7.1 col Namespace Reference

Collision detection namespace.

Classes

- struct BoxFiller

  Helpers for Boxtree::Boxtree.

- struct compElemByCenter

  Compare two elemntary boxes by the center point along one axis.

- struct compElemByMin

  Compare two elemntary boxes by the center point along one axis.

- class **Boxtree**
- class BoxtreePrecomp

  Contains all things that can be precomputed before a traversal of Boxtree's.

- class ElemBox

  Elementary box, enclosing one polygon, for Boxtree.

- class **ConvexHull**
- struct **SepPlane**
- struct **DopFiller**
- struct Dop

  A DOP is represented by NumOri (=k) many plane offsets.

- struct ElemDop

  Elementary DOP enclosing one polygon.

- struct DopTransform

  Affine transformation for DOP re-alignment.

- struct DopNode

  DOP node of the DOP hierarchy.

- class **DopTree**
- class XCollision

  Exceptions for Collision detection module.

- class **XQueueTooMany**
- class **XQueueNoLock**
- class XDopTree

  Will be raised by DopTree.

- class XColBug

  Will be raised by collision detection module, if a bug occurs somewhere in the code.

- class XBoxtree

  Will be raised by BoxTree.

- class **Grid**
- struct **GridObjLtstr**
- class **GridCell**
- class **GridObj**
- class **VtableTest_Pnt3f**
- class **VtableTest_Vec3f**
- class **VtableTest1**
- class **VtableTest2**
- struct Callback

  This is a functor for collision callbacks.

- struct **PolygonIntersectionData**
- struct Data

  Holds results from collision detection and client data.

- class CollisionPipeline

  This implements the whole collision detection pipeline, from front-end over broad-phase(s) to narrow-phase.

- struct SyncFun

  This is a functor for synchronization with other threads.

- class ColObj

  One collidable object.

- class ColPair

  Pairs of ColPObjs's.

- class MatrixCell

  A single cell of the collision interest matrix.

- class Matrix

  The collision interest matrix.

- struct ColPipelineData

  Struct to store some things which are used in Collision Detection Pipeline.

- class **Queue**
- struct Request

  Each request from the application is encapsulated by an instance of this class.

- struct **EquivPoint**
- struct TopoFace

  A face is a sorted array of indices into some vertrex array.

- class Topology

  Zur Beschreibung von Inzidenz- und Adjazenz-Relationen.

- class **VertexIterator**
- struct lessByAngle

  Compare points by angle.

- struct sBF

  contains some state across different invocations of addFace()

- class NanoTimer

  Timer with nanoseconds resolution.

- class FibRand

  Lagged Fibonacci random sequence.

- class **VisDebug**

Typedefs

- typedef std::vector< const DopNode * > **DopNodeList**
- typedef bool(*) PolyIntersectT (Data *data)

  User-provided function for intersecting a pair of polygons.

Enumerations

- enum LevelOfDetectionE { **LEVEL_BOX**, **LEVEL_HULL**, **LEVEL_EXACT** }

  Detection levels for Callback.

- enum AlgoE { ALGO_DEFAULT, **ALGO_DOPTREE**, **ALGO_BOXTREE** }

  Algorithm to apply for rigid collision detection.

- enum RequestE {
  ADD_OBJECT, **ADD_CALLBACK**, **REMOVE_CALLBACK**, **ACTIVATE_OBJECT**,
  DEACTIVATE_OBJECT, **ADD_CYCLE_CALLBACK** }

  The types of requests (besides check()) to the collision detection module.

Functions

- **BOOST_STATIC_ASSERT** (Boxtree::M_MaxNVertices==4)
- bool intersectPolygons (const Pnt3f *poly1, int plSize1, const Pnt3f *poly2, int plSize2, const unsigned int *index1, const unsigned int *index2, const osg::Matrix *cxform)

  Check if two polygons intersect.

- bool intersectQuadrangles (const osg::Pnt3f &polyVv0, const osg::Pnt3f &polyVv1, const osg::Pnt3f &polyVv2, const osg::Pnt3f &polyVv3, const osg::Pnt3f &polyUv0, const osg::Pnt3f &polyUv1, const osg::Pnt3f &polyUv2, const osg::Pnt3f &polyUv3, const osg::Vec3f &normal1V, const osg::Vec3f &normal2V)

  Checks whether two quadrangles intersect.

- bool intersectTriangles (const Pnt3f &polyVv0, const Pnt3f &polyVv1, const Pnt3f &polyVv2, const Pnt3f &polyUv0, const Pnt3f &polyUv1, const Pnt3f &polyUv2)

  Checks if two triangles intersect.

- bool intersectTriangles (const Pnt3f &polyVv0, const Pnt3f &polyVv1, const Pnt3f &polyVv2, const Pnt3f &polyUv0, const Pnt3f &polyUv1, const Pnt3f &polyUv2, const Vec3f &n1V, const Vec3f &n2V)

  Checks if two triangles intersect.

- bool intersectCoplanarEdges (const Pnt3f &v0V, const Pnt3f &v1V, const Pnt3f &u0V, const Pnt3f &u1V, unsigned int x, unsigned int y)

  Checks if the edges intersect in 2D.

- bool intersectEdgePolygon (const Pnt3f &v1, const Pnt3f &v2, const Pnt3f *poly, unsigned int plSize, const Vec3f &normalV, unsigned int x, unsigned int y)

  Checks, if edge intersects polygon.

- bool intersectEdgePolygon (const Pnt3f &v1, const Pnt3f &v2, const Pnt3f *poly, unsigned int plSize)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- bool intersectArbPolygons (const Pnt3f *poly1, unsigned int plSize1, const Pnt3f *poly2, unsigned int plSize2, const Vec3f &normal1V, const Vec3f &normal2V)

  Checks if two polygons intersect.

- bool intersectArbPolygons (const Pnt3f *poly1, unsigned int plSize1, const Pnt3f *poly2, unsigned int plSize2)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- bool **intersectPolygons** (const osg::Pnt3f *poly1, int plSize1, const osg::Pnt3f *poly2, int plSize2, const unsigned int *index1=NULL, const unsigned int *index2=NULL, const osg::Matrix *cxform=NULL)
- bool **intersectCoplanarTriangles** (const osg::Vec3f &normalV, const osg::Pnt3f &polyVv0, const osg::Pnt3f &polyVv1, const osg::Pnt3f &polyVv2, const osg::Pnt3f &polyUv0, const osg::Pnt3f &polyUv1, const osg::Pnt3f &polyUv2)
- bool **intersectTriangles** (const osg::Pnt3f &polyVv0, const osg::Pnt3f &polyVv1, const osg::Pnt3f &polyVv2, const osg::Pnt3f &polyUv0, const osg::Pnt3f &polyUv1, const osg::Pnt3f &polyUv2)

- bool **intersectTriangles** (const osg::Pnt3f &polyVv0, const osg::Pnt3f &polyVv1, const osg::Pnt3f &polyVv2, const osg::Pnt3f &polyUv0, const osg::Pnt3f &polyUv1, const osg::Pnt3f &polyUv2, const osg::Vec3f &n1, const osg::Vec3f &n2)
- bool **intersectEdgePolygon** (const osg::Pnt3f &v1, const osg::Pnt3f &v2, const osg::Pnt3f *poly, int c, const osg::Pnt3f &normalV, unsigned int x, unsigned int y)
- bool **intersectEdgePolygon** (const osg::Pnt3f &v1, const osg::Pnt3f &v2, const osg::Pnt3f *poly, unsigned int plSize)
- bool **intersectArbPolygons** (const osg::Pnt3f *poly1, unsigned int plSize1, const osg::Pnt3f *poly2, unsigned int plSize2)
- bool **intersectArbPolygons** (const osg::Pnt3f *poly1, unsigned int plSize1, const osg::Pnt3f *poly2, unsigned int plSize2, const osg::Vec3f &normal1V, const osg::Vec3f &normal2V)
- bool **intersectCoplanarEdges** (const osg::Pnt3f &v0V, const osg::Pnt3f &v1V, const osg::Pnt3f &u0V, const osg::Pnt3f &u1V, unsigned int x, unsigned int y)
- **BOOST_STATIC_ASSERT** (sizeof(VtableTest1)==sizeof(VtableTest2))
- **BOOST_STATIC_ASSERT** (sizeof(osg::Pnt3f)!=sizeof(VtableTest_Pnt3f))
- **BOOST_STATIC_ASSERT** (sizeof(osg::Vec3f)!=sizeof(VtableTest_Vec3f))
- float **operator** * (const Vec4f &vec4, const Pnt3f &pnt3)
- Pnt3f **barycenter** (const MFPnt3f *points, const unsigned int index[ ], const unsigned int nindices)
- void **getNodeBBox** (NodePtr node, float min[3], float max[3])
- GeometryPtr **getGeom** (const NodePtr node)
- MFPnt3f * **getPoints** (const NodePtr node)
- MFPnt3f * **getPoints** (const GeometryPtr geo)
- void **mergeGeom** (const NodePtr &subtree, NodePtr *geonode)
- void **mlerp** (osg::Matrix *intermat, const osg::Matrix &m1, const osg::Matrix &m2, float t)
- unsigned int **sign** (double &x)
- unsigned int **sign** (int x)

Creation, desctruction, assignments

Constructors and desctructors

- void **fillDops** (const osg::NodePtr &node, const osg::GeometryPtr &geo, const osg::FaceIterator &fi, void *data)

Vector, Matrix, and Transformation Math

- float operator * (const Vec3f &vec3, const Vec4f &vec4)

  *Several 'vector * vector' and 'vector * point' products.*

- float operator * (const Pnt3f &pnt, const float vec[3])

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- float operator * (const osg::Vec4f &vec4, const Pnt3f &pnt3)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- float operator * (const Pnt3f &pnt3, const Vec3f &vec3)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- void operator+= (Vec4f &vec4, const Vec3f &vec3)

Vec4f += Vec3f.

- Pnt3f lincomb (float c1, const Pnt3f &pnt1, float c2, const Pnt3f &pnt2)
  Affine combination of two points.
- void getTransfomUpto (const osg::NodePtr &cur, const osg::NodePtr &upto, osg::Matrix &result)
  Combine all transformation matrices between two nodes in the graph.
- void iterFaces (const osg::NodePtr &node, void(*callback)(const osg::NodePtr &, const osg::GeometryPtr &, const osg::FaceIterator &, void *), void *data)
  Calls a function for every face in the scenegraph.
- void countFaces (const osg::NodePtr &, const osg::GeometryPtr &, const osg::FaceIterator &, void *data)
  Count the number of faces in a scenegraph.
- float dist2 (const Pnt3f &pnt1, const Pnt3f &pnt2)
  Square distance between 2 points.
- float dist (const Pnt3f &pnt1, const Pnt3f &pnt2)
  Distance between 2 points.
- Pnt3f barycenter (const Pnt3f *points, const unsigned int npoints)
  Average of an array of points.
- Pnt3f barycenter (const vector< Pnt3f > &points)
  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- Pnt3f barycenter (const Pnt3f *points, const unsigned int index[ ], const unsigned int nindices)
  Average of an array of indexed points.
- Pnt3f barycenter (const osg::MFPnt3f *points, const unsigned int index[ ], const unsigned int nindices)
  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- Pnt3f barycenter (const vector< Pnt3f > &points, const TopoFace &face)
  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- bool collinear (const Vec3f &a, const Vec3f &b)
  Test if two vectors are collinear.
- bool coplanar (const Pnt3f &p0, const Pnt3f &p1, const Pnt3f &p2, const Pnt3f &q0, const Pnt3f &q1, const Pnt3f &q2)
  Test if two triangles (planes / polygons) are coplanar.
- Vec3f operator * (const osg::Matrix &m, const Vec3f &v)
  *Matrix * Vec3f.*
- Pnt3f mulM3Pnt (const osg::Matrix &m, const Pnt3f &p)
  *Matrix * Pnt3f.*

- Pnt3f operator ∗ (const osg::Matrix &m, const Pnt3f &p)
  Matrix ∗ vector.
- osg::Matrix operator ∗ (const osg::Matrix &m1, const osg::Matrix &m2)
  Matrix ∗ matrix.
- Vec3f mulMTVec (const osg::Matrix &m, const Vec3f &v)
  Transposed matrix ∗ Vec3f.
- void printMat (const osg::Matrix &m, FILE ∗file)
  Print a matrix.
- void printPnt (const osg::Pnt3f &p, FILE ∗file)
  Print a point.
- void dominantIndices (const Vec3f &v, unsigned int ∗x, unsigned int ∗y)
  Dominant coord plane which v is "most orthogonal" to.
- void dominantIndices (const Vec3f &v, unsigned int ∗x, unsigned int ∗y, unsigned int ∗z)
  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- unsigned int dominantIndex (const Vec3f &v)
  Dominant coord axis which v is "most parallel" to.
- Vec3f triangleNormal (const Pnt3f &p0, const Pnt3f &p1, const Pnt3f &p2)
  Normal of a triangle defined by 3 points.
- osg::Matrix axisToMat (const Vec3f &a, float d)
  Convert a rotation given by axis & angle to a matrix.
- unsigned int discretizeOri (osg::Quaternion q, unsigned int r)
  Convert an orientation (quarternion) into an integer (e.g., index).
- void mlerp (OSG::Matrix ∗intermat, const OSG::Matrix &m1, const OSG::Matrix &m2, float t)
  Matrix linear interpolation.

Geometry

- void sortVerticesCounterClockwise (const vector< Pnt3f > &vertex, const Vec3f &normal, TopoFace &face)
  Sort vertices of a face such that they occur counter clockwise.
- osg::NodePtr geomFromPoints (const vector< Pnt3f > &vertex, vector< TopoFace > &face, int gl_type, bool skip_redundant, const Vec3f normals[ ])
  Create a polyhedron from simple vertex and face arrays.
- osg::NodePtr geomFromPoints (const Pnt3f vertex[ ], unsigned int nvertices, unsigned int face[ ], const unsigned int face_nv[ ], unsigned int nfaces, int gl_type, bool skip_redundant, const Vec3f normals[ ])
  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- osg::NodePtr makeCube (float radius, int gl_type)

Create a cube as OpenSG object.

- void getNodeBBox (osg::NodePtr node, float min[3], float max[3])
  Get BoundingBox of an osg-node.
- osg::GeometryPtr getGeom (const osg::NodePtr node)
  Return the pointer to the geometry core of the node.
- osg::MFPnt3f * getPoints (const osg::NodePtr node)
  Return the pointer to the multi-field of the points.
- osg::MFPnt3f * getPoints (const osg::GeometryPtr geo)
  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- osg::GeoPositions3fPtr getPositions (const osg::NodePtr node)
  Return the GeoPositionsPtr of a node.
- void calcVertexNormals (const osg::NodePtr node, const float creaseAngle)
  Calculate vertex normals for all geometries in a subtree.
- osg::NodePtr findGeomNode (const osg::NodePtr node)
  Find the first node that has a geometry.
- osg::MaterialPtr findMaterial (const osg::NodePtr node)
  Return the material a geometry node is being drawn with.
- void addFace (const osg::NodePtr &node, const osg::GeometryPtr &geo, const osg::FaceIterator &face, sBF *bf)
  Add one face to a geometry/node; used by addAllFaces.
- void addAllFaces (const osg::NodePtr &root, sBF *bf)
  Copy all faces in the subtree into one geometry; used by mergeGeom().
- void mergeGeom (const osg::NodePtr &subtree, osg::NodePtr *geonode)
  Merge all geometries in a subtree into a node.

Timers, timing, sleeping, etc.

- void sleep (unsigned int microseconds)
  Sleep n microseconds.
- float time (void)
  Get the user time in milliseconds.

Random numbers

- double my_drand48 (void)
  Substitute for the drand48() function under Unix (needed under Windoze).
- unsigned int pseudo_random (void)
  Pseudo random number generator.

- float pseudo_randomf (void)
  Pseudo random number generator.

Floating-Point Tricks

- unsigned int sign (float &x)
  Returns 0 if x < 0, 0x80000000 otherwise.

Misc

- bool lockToProcessor (unsigned int processor)
  Lock the calling process to a certain processor.

Intersection Tests

- bool isectCoplanarTriangles (const Vec3f &normalV, const Pnt3f &polyVv0, const Pnt3f &polyVv1, const Pnt3f &polyVv2, const Pnt3f &polyUv0, const Pnt3f &polyUv1, const Pnt3f &polyUv2)
  Checks whether two coplanar triangles intersect.
- bool isectCoplanarEdges (const Pnt3f &v0V, const Pnt3f &v1V, const Pnt3f &u0V, const Pnt3f &u1V, unsigned int x, unsigned int y)
  Checks if the edges intersect in 2D.
- void isectEdgePolygon (const Pnt3f &v1, const Pnt3f &v2, const Pnt3f *poly, unsigned int plSize, const Vec3f &normalV, unsigned int x, unsigned int y, bool *isect, bool *oneside)
  Checks, if edge intersects polygon in 2D.
- void **isectEdgeTriangle** (const Pnt3f &v1, const Pnt3f &v2, const Pnt3f *poly, const Vec3f &normalV, unsigned int x, unsigned int y, bool *isect, bool *oneside)
- bool pointInPolygon (const Pnt3f &pt, const Pnt3f *poly, unsigned int plSize, unsigned int x, unsigned int y)
  Check if point is inside polygon.
- bool pointInTriangle (const Pnt3f &pt, const Pnt3f &v0, const Pnt3f &v1, const Pnt3f &v2, unsigned int x, unsigned int y)
  Check whether point is inside triangle.

Variables

- const float M_InitEta = 0.1
  Some constants for the separating planes algo ConvexHull::check(); optimal values determined by experiments.
- const float **M_MaxSteps** = 150
- const float **M_AnnealingFactor** = 0.97
- int **vvv**
- const unsigned int MaxNVertices = 10
  Maximal number of vertices a polygon is allowed to contain.

- const float NearZero = 1E-6

  Epsilon; all collision detection math will use this threshold.

7.1.1 Detailed Description

Collision detection namespace.

7.1.2 Typedef Documentation

7.1.2.1 typedef bool(∗) col::PolyIntersectT(Data ∗data)

User-provided function for intersecting a pair of polygons.

Parameters:

data contains various info about the pair of objects and the pair of polygons to be checked

The user can provide her own function for intersecting polygons. Whenever a collision detection algorithm has to determine the intersection status of a pair of polygons, it will call this function. Whether or not the application program really checks the intersection is up to the application programmer; it could be used for other things like coloring the polygons.

data->polisecdata->pgon[0] is guaranteed to be a member of data->geom[0], and data->polisecdata->pgon[1] is part of data->geom[1].

Author:

Gabriel Zachmann

Todo

Als Funktor machen!

7.1.3 Enumeration Type Documentation

7.1.3.1 enum col::AlgoE

Algorithm to apply for rigid collision detection.

Enumerator:

ALGO_DEFAULT this is usually the best

7.1.3.2 enum col::RequestE

The types of requests (besides check()) to the collision detection module.

Warning:

If you change this, you *must* change Request::Names!

7.1.4 Function Documentation

7.1.4.1 void col::addAllFaces (const osg::NodePtr & *root*, sBF * *bf*)

Copy all faces in the subtree into one geometry; used by mergeGeom().

Parameters:

root root of the subtree

bf contains state; must be init'ed by caller

7.1.4.2 void col::addFace (const osg::NodePtr & *node*, const osg::GeometryPtr & *geo*, const osg::FaceIterator & *face*, sBF * *bf*)

Add one face to a geometry/node; used by addAllFaces.

Parameters:

node/geo the node / geometry to which the *face* is added

face points to the face to be added

bf contains state across successive invocations

Implementation:

Only for internal usage, probably.

7.1.4.3 osg::Matrix col::axisToMat (const Vec3f & *a*, float *d*)

Convert a rotation given by axis & angle to a matrix.

Parameters:

a axis (unit vector)

d angle (radians)

Returns:

m rotation matrix

Precondition:

- The matrix will be used via "vector * matrix".
- The axis must be a unit vector.

The axis/angle have the meaning: "rotate about the axis with angle deg" The right-hand rule is used.

The matrix will be of the form $\begin{pmatrix} & & & 0 \\ & \text{Rot} & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ (Source: W. Schindler)

Implementation:

Ported from Y/conversion.c

Todo

d = -d; kann man wahrscheinlich wieder rauswerfen, wenn man unten die Transposition auch entfernt.

7.1.4.4 Pnt3f col::barycenter (const Pnt3f * *points*, const unsigned int *index*[ ], const unsigned int *nindices*)

Average of an array of indexed points.

Parameters:

points the array

index,nindices array of indices into points

Returns:

The average over all points[index[i]], i = 0 .. nindices-1.

Warning:

No range check on indices will done, i.e., indices pointing outside points[] will produce garbage or an FPE!

7.1.4.5 Pnt3f col::barycenter (const Pnt3f * *points*, const unsigned int *npoints*)

Average of an array of points.

Parameters:

points the array

npoints number of points

Returns:

The average.

7.1.4.6 void col::calcVertexNormals (const osg::NodePtr *node*, const float *creaseAngle* = 90.0)

Calculate vertex normals for all geometries in a subtree.

Parameters:

node root of subtree to be processed

creaseAngle dihedral(?) angles larger than this won't be averaged (degrees)

7.1.4.7 bool col::collinear (const Vec3f & *a*, const Vec3f & *b*)

Test if two vectors are collinear.

Parameters:

a,b the vectors

Returns:

true if collinear, false otherwise.

Check wether or not a = l ∗ b, l!=0. A 0 vector is not considered collinear with any other vector (if both a and b are 0, they are still considered *not* collinear).

This is (hopefully) only a temporary function, until available from OSG.

7.1.4.8 bool col::coplanar (const Pnt3f & *p0*, const Pnt3f & *p1*, const Pnt3f & *p2*, const Pnt3f & *q0*, const Pnt3f & *q1*, const Pnt3f & *q2*)

Test if two triangles (planes / polygons) are coplanar.

Parameters:

p0,p1,p2 first triangle / plane / polygon

q0,q1,q2 second ...

Returns:

true if colplanar, false otherwise.

Check wether the two planes given by the 2x3 sets of points are coplanar. If the two sets of points are from two different polygons, then this function returns true, if both polygons lie in the same plane.

Precondition:

At least one of the two triangles should yield a normal unequal 0.

Warning:

Not optimized.

7.1.4.9 void col::countFaces (const osg::NodePtr &, const osg::GeometryPtr &, const osg::FaceIterator &, void ∗ *data*)

Count the number of faces in a scenegraph.

Warning:

Don't call it directly, use iterFaces!

7.1.4.10 unsigned int col::discretizeOri (osg::Quaternion *q*, unsigned int *r*)

Convert an orientation (quarternion) into an integer (e.g., index).

Parameters:

q quaternion

r resolution of the discretization (must be > 0, and even)

Returns:

The integer representing the quaternion. The range is 0, .., 6∗r∗r∗(r/2) + r∗r∗r = $4∗r^3-1$.

Discretizes the space of all rotations (= S^3), and returns a unique integer for each rotations belonging to the same equivalence class w.r.t. this discretization.

q and -q should return the same integer.

Note that if the angle is zero (q[3]==1), you still get different indices, depending on the axis, although the rotation is always the same, namely the identity.

Note also, that bogus quaternions with a zero axis (0,0,0,a) will still produce an index within the range.

So, the range of indices produced by this function over all "sensible" unit quaternions does not cover all of [0,$4∗r^3-1$].

The discretization is done by rastering the 4-dim. unit circumcube.

Exceptions:

XCollision If r==0 or |q|<eps .

XColBug If there is an internal bug; shouldn't happen.

Warning:

If *r* is not even, then it will be incremented. The quaternion *q must* have unit length - otherwise bogus will be returned.

Bug

I think, that if two rotations yield the same index, then they represent "close" rotations - but I haven't checked yet. (Note that the reverse statement is not true.)

See also:

...

Implementation:

The quaternion is mirrored (q=-q), if q[4]<0, so as to stay on the upper hemisphere. Then, q is projected on the surrounding unit hemicube ($[-1,1]^4$). There are 7 sides. The sides of that hemicube are superimposed with a raster: the "top" side has r∗r∗r squares, all other sides have r∗r∗(r/2) squares.

Special care has been taken to make sure that quaternions, which are projected exactly on a hemicube edge (2 or more q[i] = 1), are consistently turned into an index.

Remember that the faces of a 4-dim. cube are 3-dim cubes.

7.1.4.11 float col::dist (const Pnt3f & *pnt1*, const Pnt3f & *pnt2*)

Distance between 2 points.

Parameters:

pnt1 point
pnt2 point

Returns:

The distance.

This is (hopefully) only a temporary function, until available from OSG.

7.1.4.12 float col::dist2 (const Pnt3f & *pnt1*, const Pnt3f & *pnt2*)

Square distance between 2 points.

Parameters:

pnt1 point
pnt2 point

Returns:

The squared distance.

This is (hopefully) only a temporary function, until available from OSG.

7.1.4.13 unsigned int col::dominantIndex (const Vec3f & *v*)

Dominant coord axis which v is "most parallel" to.

Parameters:

v vector

Returns:

Index of maximum coordinate of *v*, such that $v_x \geq \max(v_i)$.

7.1.4.14 void col::dominantIndices (const Vec3f & *v*, unsigned int * *x*, unsigned int * *y*, unsigned int * *z*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

v Vector
x
y
z index of smallest vector coord (out)

7.1.4.15 void col::dominantIndices (const Vec3f & *v*, unsigned int * *x*, unsigned int * *y*)

Dominant coord plane which v is "most orthogonal" to.

Parameters:

v vector

x,y indices of "most orthogonal" plane (out)

Compute *x* and *y*, such that $\min(v_i) \leq v_x \ \wedge \ \min(v_i) \leq v_y$.

7.1.4.16 osg::NodePtr col::findGeomNode (const osg::NodePtr *node*)

Find the first node that has a geometry.

Parameters:

node root of subtree to be searched

Returns:

The node having a geometry, or osg::NullFC.

Make a depth-first traversal of the subtree starting at *node*, and return the first node that has a geometry.

7.1.4.17 osg::MaterialPtr col::findMaterial (const osg::NodePtr *node*)

Return the material a geometry node is being drawn with.

Parameters:

node root of subtree to be searched

Returns:

The material a geometry node is being drawn with. Can return osg::NullFC in 2 cases: *node* does not have a geometry core, or, findMaterial doesn't find a material on the path from the root to *node*.

7.1.4.18 osg::NodePtr col::geomFromPoints (const Pnt3f *vertex*[ ], unsigned int *nvertices*, unsigned int *face*[ ], const unsigned int *face_nv*[ ], unsigned int *nfaces*, int *gl_type*, bool *skip_redundant*, const Vec3f *normals*[ ])

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

vertex,nvertices vertex array (in)

face,face_nv,nfaces faces array (in, could get sorted)

gl_type = GL_LINE_LOOP, GL_POLYGON, etc...

skip_redundant skip faces with <3 vertices, if true

normals normals array (can be NULL)

Face_nv[i] contains the number of vertices of *face*[i]. *Nfaces* contains the number of faces in *face*. There may be redundant faces, i.e., faces with face_nv[i] = 0,1,2.

Exceptions:

XColBug If a face has more than NumOri many vertices.

Precondition:

- *normals* has *nfaces* many normals.
- *face*[i] *must* have *Dop::NumOri* many columns!

Todo

- immer noch wird die Variable NumOri gebraucht..

7.1.4.19 osg::NodePtr col::geomFromPoints (const vector< Pnt3f > & *vertex*, vector< TopoFace > & *face*, int *gl_type*, bool *skip_redundant*, const Vec3f *normals*[ ])

Create a polyhedron from simple vertex and face arrays.

Parameters:

vertex vertex array (in)

face faces array (in, could get sorted)

gl_type = GL_LINE_LOOP, GL_POLYGON, etc...

skip_redundant skip faces with <3 vertices, if true

normals normals array (can be NULL)

Returns:

The node.

Create a polyhedron which is the object described by the planes given by normals, nfaces, vertex, face, face_nv. Actually, normals is only used for sorting the vertices of the faces. If *normals* == NULL, then we assume that vertices in *face*[] are already sorted in counter-clockwise order. *Face* contains indices into vertex.

There may be redundant faces, i.e., faces with face[i].size() = 0,1,2. The geometry will have no material.

No material is assigned.

Warning:

If *normals* != NULL, then *face will* get sorted!

Exceptions:

XColBug If there are no faces with any vertices.

Precondition:

- Planes are given by Ori∗x - d = 0.
- *normals* has *face.size()* many normals.

7.1.4.20 osg::GeometryPtr col::getGeom (const osg::NodePtr *node*)

Return the pointer to the geometry core of the node.

Exceptions:

XCollision If there is no geometry, i.e., if the dcast failed

7.1.4.21 void col::getNodeBBox (osg::NodePtr *node*, float *min*[3], float *max*[3])

Get BoundingBox of an osg-node.

Parameters:

node the Inpute-Node

min,max the BBox

7.1.4.22 osg::MFPnt3f* col::getPoints (const osg::NodePtr *node*)

Return the pointer to the multi-field of the points.

Warning:

Don't use the MFPnt3f pointer to modify the geometry! OpenSG will not notice the changes!

Exceptions:

XCollision If any of the `dynamic_cast's` returns `NULL`.

7.1.4.23 osg::GeoPositions3fPtr col::getPositions (const osg::NodePtr *node*)

Return the GeoPositionsPtr of a node.

Exceptions:

XCollision If any of the `dynamic_cast's` returns `NULL`.

7.1.4.24 void col::getTransfomUpto (const osg::NodePtr & *cur*, const osg::NodePtr & *upto*, osg::Matrix & *result*)

Combine all transformation matrices between two nodes in the graph.

Parameters:

cur lowest node of the graph

upto highest node in the graph

result the resulting transformation matrix

7.1.4.25 bool col::intersectArbPolygons (const Pnt3f * *poly1*, unsigned int *plSize1*, const Pnt3f * *poly2*, unsigned int *plSize2*, const Vec3f & *normal1V*, const Vec3f & *normal2V*)

Checks if two polygons intersect.

Parameters:

poly1 the vertices of the first polygon

poly2 the vertices of the second polygon

normal1V,normal2V normals of polygons (optional)

Returns:

true, if the polygons intersect, false, otherwise

Checks if the two polygons intersect using intersectEdgePolygon Normals are calculated if not provided as parameters.

Warning:

If you change this function, check whether the other two polygon- functions must also be changed!!

Precondition:

poly1 and *poly2* contain at least 3 vertices and are not degenerated. The vertices must be in consistent order.

7.1.4.26 bool col::intersectCoplanarEdges (const Pnt3f & *v0V*, const Pnt3f & *v1V*, const Pnt3f & *u0V*, const Pnt3f & *u1V*, unsigned int *x*, unsigned int *y*)

Checks if the edges intersect in 2D.

Parameters:

v0V,v1V vertices of first edge

u0V,u1V vertices of second edge

x,y indices (in {0,1,2}) to dominant plane

Returns:

true if the edges intersect false otherwise

This edge to edge test is based on Franlin Antonio's gem: "Faster Line Segment Intersection" in Graphics Gems III, pp. 199-202.

Precondition:

v0, *v1*, *u0* and *u1* describe valid non-degenerated line segments. Both line segments are coplanar.

Todo

Optimierung: Faktorisieren, um erste zwei Berechungen nicht mehrfach mit gleichen Parametern durchzufuehren!!!

7.1.4.27 bool col::intersectEdgePolygon (const Pnt3f & *v1*, const Pnt3f & *v2*, const Pnt3f * *poly*, unsigned int *plSize*, const Vec3f & *normalV*, unsigned int *x*, unsigned int *y*)

Checks, if edge intersects polygon.

Parameters:

v1,v2 vertices of a line segment edge

poly vertices of an arbitrary polygon

normalV normal (optional)

x,y indices (in {0,1,2}) to dominant plane (optional)

Returns:

true, if the edge intersects the polygon, false, otherwise

Checks, if the edge intersects the polygon using an algorithm originally implemented in arbcoll.c using the Y library. The algorithm has been ported to the Cosmo 3D library. See function edgePolygon in arbcoll.c

The original function did not handle the coplanar case, this implementation does.

Precondition:

v1 and *v2* describe a valid line segment, *poly* contains at least 3 elements, the elements in *poly* are all in the same plane and define a valid polygon.

Todo

Schleife ueber intersectCoplanarEdges kann optimiert werden!

7.1.4.28 bool col::intersectPolygons (const Pnt3f * *poly1*, int *plSize1*, const Pnt3f * *poly2*, int *plSize2*, const unsigned int * *index1*, const unsigned int * *index2*, const osg::Matrix * *cxform*)

Check if two polygons intersect.

Parameters:

poly1,poly2 vertices of the first,second polygon

plSize1,plSize2 number of vertices of first,second polygon; may be -4 to indicate a quadstrip polygon!

index1,index2 index arrays into poly1, poly2 (possibly = NULL)

cxform coordinate transformation to transform first polygon into frame of second polygon

Returns:

true, if the polygons intersect, false, otherwise

Check if the two polygons intersect by using either the triangle intersection test by Tomas Moeller or the edge against polygon test, whichever is faster. Normals are calculated always.

If the index arrays index1/2 are set, then the vertices of the polygon are poly[index[0]], .. poly[index[plSize-1]]; otherwise, the vertices are poly[0], .., poly[plSize-1].

The edges of quadrangles can be ordered in two ways, which are distinguished by the sign of *plSize1/2* (4 corresponds to a normal quadrangle, -4 to a quadstrip).

```
quadrangle          quadstrip
   1--2                1--3
   | /|                | /|
   |/ |                |/ |
   0--3                0--2
```

If possible, pass that polygon as first parameter, which has less vertices.

Exceptions:

XCollision Falls *abs*(plSize1/2) > MaxNVertices.

Todo

Da ein Viereck planar ist, braucht man eigentlich die Unterscheidung zwischen Quadrangle und Quadstrip doch nicht machen, oder?

Warning:

- The implementation assumes, that Pnt3f has no virtual function table!!
- If *plSize1/2* < 0 and *plSize* != -4, then it will probably dump core; this is not checked, so as to retain performance.

Precondition:

poly1 and *poly2* contain >= 3 vertices and are not degenerated. Polygons must be convex and planar. The vertices must be in consistent order (clockwise or counter-clockwise).

Implementation:

If you change this function, check whether the other two polygon- functions must also be changed!

Implementation:

Because of the index option (`index1/2`), I have to copy vertices under certain circumstances, which *might* be a little performance hit. On the other hand, if I wanted to avoid that, I would have to duplicate code ...

7.1.4.29 bool col::intersectQuadrangles (const osg::Pnt3f & *polyVv0*, const osg::Pnt3f & *polyVv1*, const osg::Pnt3f & *polyVv2*, const osg::Pnt3f & *polyVv3*, const osg::Pnt3f & *polyUv0*, const osg::Pnt3f & *polyUv1*, const osg::Pnt3f & *polyUv2*, const osg::Pnt3f & *polyUv3*, const osg::Vec3f & *normal1V*, const osg::Vec3f & *normal2V*)

Checks whether two quadrangles intersect.

Parameters:

polyVv0,..,polyVv3 vertices of first quadrangle (called 'V')

polyUv0,..,polyUv3 vertices of second quadrangle (called 'U')

Returns:

true, if the triangles intersect, false otherwise

Precondition:

Both quadrangles are ordered as shown (no quadstrip!)

```
1--2
| /|
|/ |
0--3
```

7.1.4.30 bool col::intersectTriangles (const Pnt3f & *polyVv0*, const Pnt3f & *polyVv1*, const Pnt3f & *polyVv2*, const Pnt3f & *polyUv0*, const Pnt3f & *polyUv1*, const Pnt3f & *polyUv2*, const Vec3f & *n1V*, const Vec3f & *n2V*)

Checks if two triangles intersect.

Parameters:

polyUv0,..,polyUv2 vertices of first triangle (called 'V')

polyVv0,..,polyVv2 vertices of second triangle (called 'U')

n1V,n2V normals for triangle V and triangle U

Returns:

true, if the triangles intersect, false otherwise.

This function is very similar to the above.

The code has been kept separate because the calculation of the normal n2V can be done after a few pre-checks which filter out a lot of triangle pairs.

Warning:

If this function is changed, make sure that you also change the overloaded funtion above!!

Precondition:

The triangles are not degenerated, i.e. all vertices are different.

7.1.4.31 bool col::intersectTriangles (const Pnt3f & *polyVv0*, const Pnt3f & *polyVv1*, const Pnt3f & *polyVv2*, const Pnt3f & *polyUv0*, const Pnt3f & *polyUv1*, const Pnt3f & *polyUv2*)

Checks if two triangles intersect.

Parameters:

polyUv0,..,polyUv2 vertices of first triangle (called 'V')

polyVv0,..,polyVv2 vertices of second triangle (called 'U')

Global Variables:

- `globalVar1` Comment for globalVar1

Returns:

true, if the triangles intersect, false otherwise.

Warning:

Dinge, die der Aufrufer unbedingt beachten muss...

Precondition:

The triangles are not degenerated, i.e. all vertices are different.

Checks, if two triangles intersect using a fast algorithm by Tomas Moeller. See article "A Fast Triangle-Triangle Intersection Test", Journal of Graphics Tools, 2 (2), 1997.

A preprocessing routine, that gets the vertices out of their surrounding structures suchs as csGeoSet might be considered useful, as this algorithm calculates the intersection using the Pnt3f data structure.

7.1.4.32 bool col::isectCoplanarEdges (const Pnt3f & *v0V*, const Pnt3f & *v1V*, const Pnt3f & *u0V*, const Pnt3f & *u1V*, unsigned int *x*, unsigned int *y*)

Checks if the edges intersect in 2D.

Parameters:

v0V,v1V vertices of first edge

u0V,u1V vertices of second edge

x,y indices (in {0,1,2}) to dominant plane

Returns:

true, if the edges intersect, false otherwise.

This edge to edge test is based on Franlin Antonio's gem: "Faster Line Segment Intersection" in Graphics Gems III, pp. 199-202.

Precondition:

v0, *v1*, *u0* and *u1* describe valid non-degenerated line segments. Both line segments are coplanar.

7.1.4.33 bool col::isectCoplanarTriangles (const Vec3f & *normalV*, const Pnt3f & *polyVv0*, const Pnt3f & *polyVv1*, const Pnt3f & *polyVv2*, const Pnt3f & *polyUv0*, const Pnt3f & *polyUv1*, const Pnt3f & *polyUv2*)

Checks whether two coplanar triangles intersect.

Parameters:

normalV normal vector of plane, in which both triangles must lie

polyVv0,..,polyVv2 vertices of first triangle (called 'V')

polyUv0,..,polyUv2 vertices of second triangle (called 'U')

Returns:

true, if the triangles intersect, false otherwise

Precondition:

The triangles are coplanar and normalV is plane's normal vector. Both triangles are not degenerated, i.e. all their vertices differ.

Checks, if the two coplanar triangles intersect. Algorithm by Tomas Moeller, see comment of intersectTriangle for details.

7.1.4.34 void col::isectEdgePolygon (const Pnt3f & *v1*, const Pnt3f & *v2*, const Pnt3f * *poly*, unsigned int *plSize*, const Vec3f & *normalV*, unsigned int *x*, unsigned int *y*, bool * *isect*, bool * *oneside*)

Checks, if edge intersects polygon in 2D.

Parameters:

v1,v2 vertices of a line segment edge

poly vertices of an arbitrary polygon

plSize size of poly

normalV normal

x,y indices (in {0,1,2}) to dominant plane

isect set if edge intersects the polygon (out)

oneside set of edge is completely on one side of the plane (out)

Checks, if the edge (v1,v2) intersects the polygon. There are three output cases:

1. isect=true is obvious;
2. isect=false and oneside=false means that the edge intersects the plane of the polygon but not the polygon itself;
3. isect=false and oneside=true means that the edge is completely on one side of the plane of the polygon.

If both edge and polygon are parallel (or coplanar), then case 2 cannot happen.

Precondition:

v1 and *v2* describe a valid line segment, *poly* contains at least 3 elements, the elements in *poly* are all in the same plane and define a valid polygon.

Todo

Schleife ueber intersectCoplanarEdges koennte optimiert werden.

Implementation:

Function edgePolygon in arbcoll.c. The original function did not handle the coplanar case, this implementation does.

7.1.4.35 void col::iterFaces (const osg::NodePtr & *node*, void(∗)(const osg::NodePtr &, const osg::GeometryPtr &, const osg::FaceIterator &, void ∗) *callback*, void ∗ *data*)

Calls a function for every face in the scenegraph.

Parameters:

node root of the graph

callback the function to call

data whatever you like

7.1.4.36 Pnt3f col::lincomb (float *c1*, const Pnt3f & *pnt1*, float *c2*, const Pnt3f & *pnt2*)

Affine combination of two points.

Parameters:

pnt1,pnt2 points

Returns:

pnt1∗*c1* + *pnt2*∗*c2*.

Precondition:

c1 + *c2* = 1!

7.1.4.37 bool col::lockToProcessor (unsigned int *processor*)

Lock the calling process to a certain processor.

Parameters:

processor the processor number

A warning is printed if the processor is not isolated (or not enabled).

Returns:

False if locking failed, true otherwise. Locking can fail if we run on a single-processor machine, or *processor* is out of bounds.

Precondition:

Processor >= 0.

Todo

Implement for Windows.

7.1.4.38 osg::NodePtr col::makeCube (float *radius*, int *gl_type*)

Create a cube as OpenSG object.

Parameters:

radius each side of the box will be 2∗size long

gl_type = GL_LINE_LOOP, GL_POLYGON, etc...

Returns:

A NodePtr to the new cube.

Creates a box with no material.

Exceptions:

XCollision See geomFromPoints().

7.1.4.39 void col::mergeGeom (const osg::NodePtr & *subtree*, osg::NodePtr ∗ *geonode*)

Merge all geometries in a subtree into a node.

Parameters:

subtree root of subtree to be searched

geonode gets all the geometry (in/out)

Make a depth-first traversal of the *subtree*, and merge all geometry cores into a new geometry core, which will then be assigned as the new core for *geonode*.

Transformations will be applied to the coordinates of the vertices, so that the new geometry looks exactly like the original *subtree*, but does not have any transformations.

Shared geometry in the *subtree* will be added multiply (possibly with different rtansformations) to *geonode*.

Warning:

Node must *not* be a node in the *subtree!*

7.1.4.40 void col::mlerp (OSG::Matrix ∗ *intermat*, const OSG::Matrix & *m1*, const OSG::Matrix & *m2*, float *t*)

Matrix linear interpolation.

Parameters:

intermat result interpolation

m1 matrix start

m2 matrix end

t 0 <= t <= 1

Calculate an in-between matrix by some sort of linear interpolation between m1 and m2. m1 and m2 should contain only scaling+rotation+translation matrices so intermat can be calc'ed correctly. Interpolates rotation, translation, ∗and∗ scalings (seperately).

I should try to interpolate those rows, which are "closest". (It's faster to use quaternions if you have to interpolate many steps.)

7.1.4.41 Pnt3f col::mulM3Pnt (const osg::Matrix & *m*, const Pnt3f & *p*)

Matrix ∗ Pnt3f.

Parameters:

m matrix

p point

Returns:

A point := m(3x3) ∗ p .

Only the upper left 3x3 part (rotation) of m is considered. This is equivalent to making the translation of m = 0.

This is (hopefully) only a temporary function, until available from OSG.

7.1.4.42 Vec3f col::mulMTVec (const osg::Matrix & *m*, const Vec3f & *v*)

Transposed matrix ∗ Vec3f.

Parameters:

m matrix

v vector

Returns:

A vector := m^T ∗ v .

Of course, only the upper left 3x3 part (rotation) of m is considered.

This is (hopefully) only a temporary function, until available from OSG.

7.1.4.43 double col::my_drand48 (void)

Substitute for the drand48() function under Unix (needed under Windoze).

Returns:

Pseudo-random number in the range [0.0 .. 1.0)

The random number is generated from rand().

7.1.4.44 Vec3f col::operator ∗ (const osg::Matrix & *m*, const Vec3f & *v*)

Matrix ∗ Vec3f.

Parameters:

m matrix

v vector

Returns:

A vector := m ∗ v .

This is (hopefully) only a temporary function, until available from OSG.

7.1.4.45 float col::operator ∗ (const Pnt3f & *pnt*, const float *vec*[3])

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

pnt point

vec vector (float array, not OSG vector)

Returns:

The dot product (i.e., projection of *pnt* on the line *vec* through origin).

7.1.4.46 float col::operator ∗ (const Vec3f & *vec3*, const Vec4f & *vec4*)

Several 'vector ∗ vector' and 'vector ∗ point' products.

Returns:

The dot product.

These operators are (hopefully) only temporary functions, until available from OpenSG.

The 4-th component of vec4 is ignored!

7.1.4.47 void col::operator+= (Vec4f & *vec4*, const Vec3f & *vec3*)

Vec4f += Vec3f.

Parameters:

vec4 4D vector

vec3 3D vector

7.1.4.48 bool col::pointInPolygon (const Pnt3f & *pt*, const Pnt3f * *poly*, unsigned int *plSize*, unsigned int *x*, unsigned int *y*)

Check if point is inside polygon.

Parameters:

pt the point to be tested (must be in plane of polygon)

x,y indices (in {0,1,2}) to dominant plane

poly polygon vertices

plSize size of poly

Returns:

true, if point is inside polygon, false otherwise.

Check if point *pt* is inside the closed polygon given by *poly*. *pt* and the vertices are 3D points, but the whole check is done with *pt* and *poly* projected onto the plane *x/y*, where *x* and *y* in {0,1,2}.

Precondition:

The vertices are assumed to define a closed polygon. pt is assumed to be in the supporting plane of the polygon.

Implementation:

This is the original pointInPolygon from Y (arbcoll.c).

Todo

Fuer Dreiecke und Vierecke optimieren!

7.1.4.49 bool col::pointInTriangle (const Pnt3f & *pt*, const Pnt3f & *v0*, const Pnt3f & *v1*, const Pnt3f & *v2*, unsigned int *x*, unsigned int *y*)

Check whether point is inside triangle.

Parameters:

pt point

v0,v1,v2 vertices of triangle

x,y indices (in {0,1,2}) to dominant plane

7.1.4.50 void col::printMat (const osg::Matrix & *m*, FILE * *file* = `stdout`)

Print a matrix.

Parameters:

m matrix

file file

Print the matrix in row-major order. This is (hopefully) only a temporary function, until available from OSG.

7.1.4.51 void col::printPnt (const osg::Pnt3f & *p*, FILE * *file* = stdout)

Print a point.

Parameters:

p the point
file output file

7.1.4.52 unsigned int col::pseudo_random (void)

Pseudo random number generator.

Returns:

A number in the range [0,2147483646].

Creates the same sequence of pseudo random numbers on every platform. Use this instead of any random(), drand48(), etc., in regression test programs.

Todo

- Check that the seed is not the unique "bad" number as explained in http://home.t-online.de/home/mok-kong.shen/ .
- *a* should be a primitive root of *c* (see URL above).
- Groesseres *c* suchen.

Bug

Not multithread-safe.

See also:

pseudo_randomf

Implementation:

Based on a linear congruential relation x(new) = a * x(old) + b (mod c). If you change *c*, you *must* change pseudo_randomf! Source: http://home.t-online.de/home/mok-kong.shen/ .

7.1.4.53 float col::pseudo_randomf (void)

Pseudo random number generator.

Returns:

A number in the range [0,1).

Creates the same sequence of pseudo random numbers on every platform. Use this instead of any random(), drand48(), etc., in regression test programs.

See also:

pseudo_random

7.1.4.54 unsigned int col::sign (float & *x*)

Returns 0 if x $<$ 0, 0x80000000 otherwise.

The test 'if (sign(x))' seems to be a little bit faster than 'if (x $<$ 0.0f)'.

For doubles and int's, use floating-point comparison instead of this function, because that's faster (see ifcomp.c).

7.1.4.55 void col::sleep (unsigned int *microseconds*)

Sleep n microseconds.

Parameters:

microseconds the sleep time

Sleeps a number of microseconds.

Under Windoze, this will sleep floor(*microseconds* / 1000) milliseconds. If this amounts to 0 milliseconds, the thread will just relinquish its time slice.

Todo

- Funktion suchen, die Mikrosekunden kann.

Bug

On most platforms (Windows, Linux, single-CPU SGI), this function will sleep at least 10 millliseconds! (On Linux, usleep and nanosleep don't work as advertised, as of RedHat 7.2)

7.1.4.56 void col::sortVerticesCounterClockwise (const vector< Pnt3f > & *vertex*, const Vec3f & *normal*, TopoFace & *face*)

Sort vertices of a face such that they occur counter clockwise.

Parameters:

vertex vertex array (in)

normal normal of face (in)

face vertex indices of the points of face (in/out)

Sort all points in face so that they will be in counterclockwise order when looked at from "outside"; "outside" is where the normal points at.

Warning:

All *face*[i] should be valid indices into *vertex!*

Precondition:

All points of *face* should lie in one plane in 3D.

Todo

Use "Lamda Library" (Boost).

Implementation:

The number of cosine evaluations is N∗log(N); it could be reduced to N.

7.1.4.57 float col::time (void)

Get the user time in milliseconds.

Returns:

Time in milliseconds.

The time is the user time of the process (and all children) since the process started.

Implementation:

Uses `times` under Unix, and `GetProcessTimes` under Windoze. Warning: Under Windoze, clock() returns wall-clock time, ∗not∗ user time! (contrary to what the MSDN documentation says!)

7.1.4.58 Vec3f col::triangleNormal (const Pnt3f & *p0*, const Pnt3f & *p1*, const Pnt3f & *p2*)

Normal of a triangle defined by 3 points.

Parameters:

p0,p1,p2 the triangle

Returns:

The normal (p1-p0) x (p2-p0).

7.1.5 Variable Documentation

7.1.5.1 const unsigned int col::MaxNVertices = 10

Maximal number of vertices a polygon is allowed to contain.

On polygons having up to MaxNVertices the intersectPolygon routines are able to perform a coordinate transformation. This maximum was introduced for performance reasons.

7.2 std Namespace Reference

STL namespace.

Classes

- class **allocator**
  STL class.
- class **auto_ptr**
  STL class.
- class **ios_base**
  STL class.
- class **basic_ios**
  STL class.
- class **basic_istream**
  STL class.
- class **basic_ostream**
  STL class.
- class **basic_iostream**
  STL class.
- class **basic_ifstream**
  STL class.
- class **basic_ofstream**
  STL class.
- class **basic_fstream**
  STL class.
- class **basic_istringstream**
  STL class.
- class **basic_ostringstream**
  STL class.
- class **basic_stringstream**
  STL class.
- class **ios**
  STL class.
- class **wios**

STL class.

- class **istream**
  STL class.
- class **wistream**
  STL class.
- class **ostream**
  STL class.
- class **wostream**
  STL class.
- class **ifstream**
  STL class.
- class **wifstream**
  STL class.
- class **ofstream**
  STL class.
- class **wofstream**
  STL class.
- class **fstream**
  STL class.
- class **wfstream**
  STL class.
- class **istringstream**
  STL class.
- class **wistringstream**
  STL class.
- class **ostringstream**
  STL class.
- class **wostringstream**
  STL class.
- class **stringstream**
  STL class.
- class **wstringstream**
  STL class.

- class **basic_string**
  STL class.
- class **string**
  STL class.
- class **wstring**
  STL class.
- class **complex**
  STL class.
- class **bitset**
  STL class.
- class **deque**
  STL class.
- class **list**
  STL class.
- class **map**
  STL class.
- class **multimap**
  STL class.
- class **set**
  STL class.
- class **multiset**
  STL class.
- class **vector**
  STL class.
- class **queue**
  STL class.
- class **priority_queue**
  STL class.
- class **stack**
  STL class.
- class **valarray**
  STL class.
- class **exception**
  STL class.

- class **bad_alloc**
  STL class.
- class **bad_cast**
  STL class.
- class **bad_typeid**
  STL class.
- class **logic_error**
  STL class.
- class **runtime_error**
  STL class.
- class **bad_exception**
  STL class.
- class **domain_error**
  STL class.
- class **invalid_argument**
  STL class.
- class **length_error**
  STL class.
- class **out_of_range**
  STL class.
- class **range_error**
  STL class.
- class **overflow_error**
  STL class.
- class **underflow_error**
  STL class.

7.2.1 Detailed Description

STL namespace.

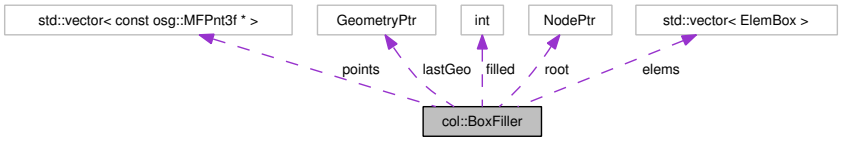
Chapter 8

CollDet Class Documentation

8.1 col::BoxFiller Struct Reference

Helpers for Boxtree::Boxtree.

Collaboration diagram for col::BoxFiller:



Public Attributes

- osg::NodePtr **root**
- vector< ElemBox > * **elems**
- vector< const osg::MFPnt3f * > * **points**
- int **filled**
- osg::GeometryPtr **lastGeo**

8.1.1 Detailed Description

Helpers for Boxtree::Boxtree.

The documentation for this struct was generated from the following file:

- ColBoxtree.cpp

8.2 Boxtree Class Reference

Implements the old axis-aligned boxtree with improvements.

8.2.1 Detailed Description

Implements the old axis-aligned boxtree with improvements.

Author:

Gabriel Zachmann, written 1997, re-implemented on OpenSG in Mar 2002.

Warning:

The destructor is *not* virtual!

See also:

For an extensive explanation of the algorithms, please see my dissertation at http://www.gabrielzachmann.org/ and the VRST'02 paper.

Todo

- Die verschiedenen *MaxNVertices* konsolidieren.

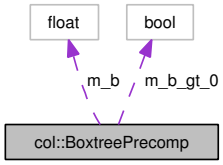
The documentation for this class was generated from the following file:

- ColBoxtree.cpp

8.3 col::BoxtreePrecomp Class Reference

Contains all things that can be precomputed before a traversal of Boxtree's.

Collaboration diagram for col::BoxtreePrecomp:



Public Member Functions

- **BoxtreePrecomp** (const osg::Matrix &m)

Public Attributes

- float m_b [3][3]

  Three unit vectors spanning the bbox of this *in* other's *coord system in Boxtree::check().*

- bool m_b_gt_0 [3][3]

  precomputed float comparisons

8.3.1 Detailed Description

Contains all things that can be precomputed before a traversal of Boxtree's.

This is a helper class for Boxtree::check() only!

Warning:

Does not work if there is a scaling or shear in *m!*

Author:

Gabriel Zachmann, written 1997, re-implemented on OSG in May 2002.

Todo

- Es ist *nicht* egal, in welches Koord.system transformiert wird! Man sollte das abhaengig von der Anzahl der Polygone machen.
- Evtl. kann man *m_b* auch einsparen.

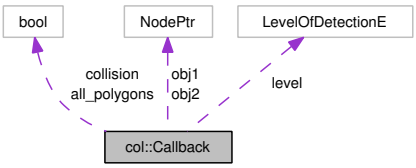
The documentation for this class was generated from the following files:

- ColBoxtree.h
- ColBoxtree.cpp

8.4 col::Callback Struct Reference

This is a functor for collision callbacks.

Collaboration diagram for col::Callback:



Public Member Functions

- virtual void operator() (const Data *data)=0 throw ()

  The raison d'etre; this will be executed by the coll. det. module.

- Callback (osg::NodePtr obj1, osg::NodePtr obj2, bool always=false, bool all_polygons_in=false, LevelOfDetectionE level_of_detection=LEVEL_EXACT)

  Create a collision callback functor.

Public Attributes

- osg::NodePtr obj1

  The two objects participating in the collision (or non-collision).

- osg::NodePtr **obj2**
- bool collision

  Tells whether or not obj1/2 have collided.

- bool all_polygons

  Tells whether or not the application is interested in all *pairs of intersecting polygons.*

- LevelOfDetectionE level

  Level of detection.

8.4.1 Detailed Description

This is a functor for collision callbacks.

Clients of the collision detection module need to derive from this abstract class and overload the () operator.

A callback can be a collision callback or a cycle callback. Collision callbacks are invoked in case of collision, cycle callbacks are invoked at the end of a completed collision cycle. Collision callbacks are only called if one or both of the objects have moved. Cycle callbacks are only called if there has been at least one object which has moved since the last cycle. Collision callbacks get some collision data (see struct Data), cycle callbacks don't.

Author:

Gabriel Zachmann

Todo

- Option vorsehen, so dass callbacks auch aufgerufen werden, wenn keines der beiden Objekte sich bewegt hat.
- Maybe we need an additional class of Callbacks, which can be re-used for several object pairs; this would just mean, that obj1/obj2 would be overwritten by the coll.det. module for every callback actually performed.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 col::Callback::Callback (osg::NodePtr *inobj1*, osg::NodePtr *inobj2*, bool *always* = `false`, bool *all_polygons_in* = `false`, LevelOfDetectionE *level_of_detection* = `LEVEL_EXACT`)

Create a collision callback functor.

Parameters:

inobj1/inobj2 callback for this pair of objs is created (both should be NullNode, if cycle callback)

all_polygons_in,always flags (see below)

level_of_detection the maximum level of detection (box, convex hull, or exact) wanted for this particular callback; the cell will later perform the max of all levels.

If *all_polygons* = true, then the col.det. module will report *all* pairs of intersecting polygons.

If *always* = true, then the col.det. module will call the callback once every collision cycle, whether the objects are colliding or not.

If the callback will be registered as a cycle callback, then both *obj1* and *obj2* should be a NullNode. Otherwise, both *obj1* and *obj2* must *not* be a NullNode; however, this will be checked only when the callback is actually saved in the collision matrix!

Todo

always flag verarbeiten.

8.4.3 Member Data Documentation

8.4.3.1 LevelOfDetectionE col::Callback::level

Level of detection.

The coll. det. module might choose to check with a finer level (for instance, if another callback for the same pair requests it).

If *all_polygons* are requested, then the finest level is assumed automatically.

The documentation for this struct was generated from the following files:

- Collision.h
- Collision.cpp

8.5 ColConvexHull Class Reference

Convex hull wrapper for qhull and collision detection of convex hulls.

8.5.1 Detailed Description

Convex hull wrapper for qhull and collision detection of convex hulls.

Implementation:

The collision detection of convex hulls is the separating planes algorithm from my thesis. See my dissertation at `http://www.gabrielzachmann.org/` .

Todo

- QHull durch CGAL ersetzen
- Den `osg::GeometryPtr` durch einen osg::GeometryConstPtr ersetzen, wenn OSG das anbietet.
- in qhull den longjmp in qh_errexit (oder so aehnlich) durch exceptions ersetzen.
- Den qhull code in einen eigenen Namespace.

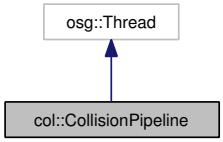
The documentation for this class was generated from the following file:

- ColConvexHull.cpp

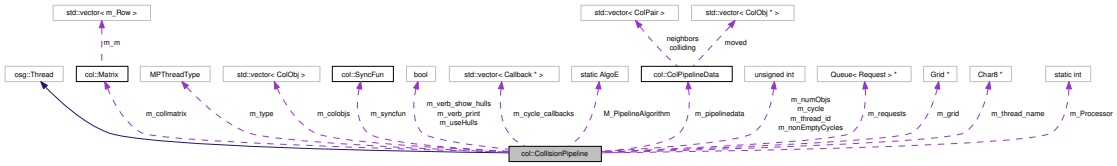
8.6 col::CollisionPipeline Class Reference

This implements the whole collision detection pipeline, from front-end over broad-phase(s) to narrow-phase.

Inheritance diagram for col::CollisionPipeline:



Collaboration diagram for col::CollisionPipeline:



Collision detection module API

- CollisionPipeline (const osg::Char8 *thread_name=NULL, unsigned int thread_id=0)

  Initialize the collision detection module.

- void check (unsigned int *num_moved=NULL)

  Collision detection query.

- void run (unsigned int number)

  If Pipeline is used as with OSGThreads, then this function starts the thread.

- void setSyncFun (SyncFun *fun)

  Set the synchronization-function.

- void useConvexHulls (bool useconvexhulls)

  Inital value is false.

- void addCallback (Callback *callback)

  Add a callback to the collision pipeline.

- void addCycleCallback (Callback *callback)

  Add a cycle-callback to the collision pipeline.

- void makeCollidable (osg::NodePtr node)

  Add an object to the collision pipeline, and set it active.

- void deactivate (osg::NodePtr node)

  Deactivate an object of the collision pipeline.

- void activate (osg::NodePtr node)

  Reactivate an object of the collision pipeline.

- unsigned int getCycle (void)

  Get current collision cycle counter.

- bool getUseGrid ()

  Returns the grid-status.

- void useGrid (unsigned int size[3], float min[3], float max[3])

  use a grid for finding neighbors

- void verbose (bool verbPrint, bool verbShowHulls)

  This is primarily for testingand debugging;.

- bool getVerbPrint ()

  The default value is false.

- bool getVerbShowHulls ()

  The default value is false.

- unsigned int getNumObjs ()

  Returns:

  the number of registered objects

- void setUseHulls (bool useHulls)

  intial value is false

- bool getUseHulls ()

  Returns:

  useHulls

- virtual ~CollisionPipeline ()

  Delete all internal structures of the collision detection pipeline.

- static CollisionPipeline * runConcurrently (char *thread_name=NULL)

  Start the collision pipeline in its own thread.

- static CollisionPipeline * get (char *name)

  Return the pipeline, if threading is used.

- static CollisionPipeline * find (char *name)

  Find a thread by its name.

- static osg::BaseThread * **create** (const osg::Char8 *thread_name, osg::UInt32 thread_id)
- virtual void workProc (void)

  Concurrent collision detection loop.

Static Public Member Functions

- static AlgoE getAlgorithm ()

  Return the algorithm which is used.

Public Attributes

- unsigned int m_thread_id

  The thread id.

- unsigned int m_nonEmptyCycles

  Collision detection loop counter.

- Queue< Request > * m_requests

  The queue for collision queries (add/remove objects).

- Matrix * m_collmatrix

  Collision interest matrix.

- std::vector< ColObj > * m_colobjs

  The list of "collidable" objects.

- std::vector< Callback * > m_cycle_callbacks

  List of cycle callbacks.

- ColPipelineData * m_pipelinedata

  Pipeline Data struct. It consist of the some data, which is used in the pipeline.

- Grid * m_grid

  3D grid

Static Public Attributes

- static AlgoE M_PipelineAlgorithm = ALGO_DOPTREE

  The algorithm to use at the back end (must be specified before check() is called).

- static int m_Processor = -1

  The number of processor on which to lock the coll.det. process.

Static Protected Attributes

- static osg::MPThreadType **m_type**

8.6.1 Detailed Description

This implements the whole collision detection pipeline, from front-end over broad-phase(s) to narrow-phase.

The idea is that you can have one instance running concurrently with the other threads of your application. (In theory, even multiple instances could be created, but this is untested.)

Exceptions:

Exception If some condition cannot be handled.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 col::CollisionPipeline::CollisionPipeline (const osg::Char8 * *thread_name* = NULL, unsigned int *thread_id* = 0)

Initialize the collision detection module.

Parameters:

thread_id Identification of the thread

thread_name Name of the thread

At compile time, we check whether or not osg::Pnt3f or osg::Vec3f have a vtable; if they do, we emit an error message, because polygonIntersect() probably won't work. I hope this check is portable.

Sideeffects:

Colmatrix, Requests, Colobjs, UseGrid, Verbose_*

Todo

- Das `useHulls` Feature vereinfachen; es gibt zu viele Stellen, wo man dieses beeinflussen kann.
- Auch das *verbose* Zeugs sollte man vielleicht aufraeumen. Vielleicht einfach durch separate Fkten machen.

8.6.3 Member Function Documentation

8.6.3.1 void col::CollisionPipeline::check (unsigned int * *num_moved* = NULL)

Collision detection query.

Parameters:

num_moved number of objects that have moved since last check (out)

This is basically the collision detection pipeline. Increment cycle counter, if some objects have moved. Swap requests queue and process them, if any. Move any objects, that have a different toWorld matrix than last cycle. Determine pairs of objects that are "neighbors" (in some sense). Filter by collision interest matrix.

Exceptions:

exception This function should not throw any exception. The idea is that exceptions are thrown only in init, construction, and other set-up functions (if any).

Warning:

If you call this function although the pipeline is running concurrently already in its own thread, then chaos will ensue!

Sideeffects:

ColObj, Verbose,

Todo

Static Variablen als Instanzvariablen der ColPipeline machen, wenn diese Funktionen hier in die Klasse ColPipeline gewandert sind. (S. Kommentar ganz oben.)

Implementation:

I use a lot of global and static arrays in order to avoid excessive ctor/dtor calls.

8.6.3.2 CollisionPipeline * col::CollisionPipeline::runConcurrently (char * *thread_name* = NULL) [static]

Start the collision pipeline in its own thread.

After this function returns, the collision pipeline will run in its own thread concurrently to all other threads.

After the thread has been created, you *must* @ not call check() any more!

Locking the new process to a certain processor work only for SGI currently. You must make sure that the processor has been isolated earlier (see `man mpadmin`).

Todo

Eigener Aspect. Sync mit anderen Threads. Gesynct werden muss eigtl. nur, wenn sich etwas an den Punkten oder Polygonen geaendert hat. Was ist, wenn Objekte geloescht wurden?

8.6.3.3 CollisionPipeline * col::CollisionPipeline::get (char * *name*) [static]

Return the pipeline, if threading is used.

Parameters:

name the name of the thread

8.6.3.4 CollisionPipeline * col::CollisionPipeline::find (char * *name*) [static]

Find a thread by its name.

Parameters:

name the name of the thread

8.6.3.5 void col::CollisionPipeline::run (unsigned int *number*)

If Pipeline is used as with OSGThreads, then this function starts the thread.

Parameters:

number the thread id

8.6.3.6 void col::CollisionPipeline::setSyncFun (SyncFun * *fun*)

Set the synchronization-function.

Parameters:

fun the synchronization-function

8.6.3.7 void col::CollisionPipeline::addCallback (Callback * *callback*)

Add a callback to the collision pipeline.

Parameters:

callback the callback-struct

8.6.3.8 void col::CollisionPipeline::addCycleCallback (Callback * *callback*)

Add a cycle-callback to the collision pipeline.

Parameters:

callback the callback-struct

8.6.3.9 void col::CollisionPipeline::makeCollidable (osg::NodePtr *node*)

Add an object to the collision pipeline, and set it active.

Parameters:

node the objetct wich should be tested for collision

8.6.3.10 void col::CollisionPipeline::deactivate (osg::NodePtr *node*)

Deactivate an object of the collision pipeline.

When an object is deactivated, it will not be tested for collisions anymore

Parameters:

node the objetct wich should be deactivated

8.6.3.11 void col::CollisionPipeline::activate (osg::NodePtr *node*)

Reactivate an object of the collision pipeline.

After an object was deactivated from collision testing, it can be reactivated with this function. It is not necessary to call this function after makeCollidable, because makeCollidable sets automatically every object as active.

Parameters:

node the objetct wich should be reactivated

8.6.3.12 unsigned int col::CollisionPipeline::getCycle (void)

Get current collision cycle counter.

Every time the collision detection module finishes a non-empty loop, this counter will be incremented. A loop is empty, if no objects have moved since the last check(). A loop finishes, when it finds out that no pairs have to be checked, or after all pairs have been checked for collision and all callbacks have been called.

The initial value of this counter is 0.

8.6.3.13 bool col::CollisionPipeline::getUseGrid ()

Returns the grid-status.

Returns:

true if grid is used

8.6.3.14 void col::CollisionPipeline::useGrid (unsigned int *size*[3], float *min*[3], float *max*[3])

use a grid for finding neighbors

If *useGrid* is not set, then pairs of "close" neighbors are determined by $O(n^2)$ bbox tests. If *useGrid* is set, then beighbors are determined by a 3-dimensional grid. However, this gains performance only, if the number of objects is very large ($>>100$) and the number of polygons per object is low. Set the Parameters for the Grid

Parameters:

size the grid will have size[0]$\sim$size[1]$\sim$size[2] cells

min,max extent of the univers

Warning:

The application *must* call this function before check()!

Points outside the grid are puted to the cell closest to them.

8.6.3.15 void col::CollisionPipeline::verbose (bool *verbPrint*, bool *verbShowHulls*)

This is primarily for testingand debugging;.

Parameters:

verbPrint some additional information as output

verbShowHulls show the hulls of the objects

8.6.3.16 bool col::CollisionPipeline::getVerbPrint ()

The default value is false.

Returns:

the status of the verbosemode for printing

8.6.3.17 bool col::CollisionPipeline::getVerbShowHulls ()

The default value is false.

Returns:

the status of the verbosemode for show hulls

8.6.3.18 void col::CollisionPipeline::setUseHulls (bool *useHulls*)

intial value is false

Parameters:

useHulls

8.6.3.19 void col::CollisionPipeline::workProc (void) [protected, virtual]

Concurrent collision detection loop.

Call check() all the time. If there is nothing to do, sleep 100 microsec.

8.6.4 Member Data Documentation

8.6.4.1 unsigned int col::CollisionPipeline::m_nonEmptyCycles

Collision detection loop counter.

This can be used by the application to determine if a new collision cycle has started.

8.6.4.2 Queue<Request>* col::CollisionPipeline::m_requests

The queue for collision queries (add/remove objects).

Implementation:

Had to make it a pointer, not a "static" global var, because we can call the ctor only after OSG has been inited.

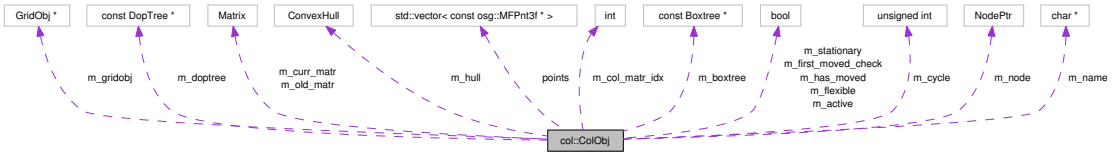
The documentation for this class was generated from the following files:

- Collision.h
- Collision.cpp

8.7 col::ColObj Class Reference

One collidable object.

Collaboration diagram for col::ColObj:



Public Member Functions

- const char * **getName** (void) const
- void setActive (bool active)

  Set the activity-status of a Colobj Used to remove an object temporarily from collision checks.

- bool isActive ()

  Returns the activity-status of a Colobj.

- void **setFlexible** (bool flexible)
- void **setStationary** (bool stationary)
- void **SetGridObj** (Grid *grid)

Constructors, desctructors, assignment

- ColObj ()

  The default ctor is not meant for "real" usage It is only there so that we can create vector's.

- ColObj (osg::GeometryPtr &geom, osg::NodePtr &node, bool flexible, bool stationary, bool compute_hull, AlgoE algo, Grid *grid, bool show_hull=false, char *name=NULL)

  The ctor you should use.

- ColObj (const ColObj &source)

  Copy a collision object.

- void operator= (const ColObj &source)

  Assignment.

Checks

- void updateBBox (void)

  Update current world bbox.

- bool bboxIntersects (ColObj *other)

  Check if the bboxes of two objects intersect.

- bool hasMoved (unsigned int global_cycle)

  Update toworld matrix and check whether or not an object has moved.

- GridObj * GetGridObj (void)
  Return GridObj.
- void updateGrid (void)
  Update GridObj if object has moved.

Static Public Member Functions

- static ColObj * find (vector< ColObj > *colobjs, osg::NodePtr obj)
  Find the colobjs*[i] for which* colobjs*[i].m_node* == obj*; If not found, returns NULL.*

Protected Attributes

- bool m_active
  Used to remove an object temporarily from collision checks.
- bool m_flexible
  Set if the object deforms.
- bool m_stationary
  Set if the object doesn't move.
- const DopTree * m_doptree
  DOP tree of this obj.
- const Boxtree * m_boxtree
  Boxtree of this obj.
- osg::NodePtr m_node
  The actual "collidable" object.
- osg::Matrix m_old_matr
  The toWorld matrix of node as of last frame.
- osg::Matrix m_curr_matr
  The current toWorld matrix as of start of current cycle.
- vector< const osg::MFPnt3f * > **points**
- char * m_name
  The name of the colobj.
- int m_col_matr_idx
  row/column index into col.
- bool m_has_moved
  Flags whether or not obj has moved since last frame.

- bool **m_first_moved_check**
- unsigned int **m_cycle**
- ConvexHull m_hull

  the convex hull of geom

- GridObj * **m_gridobj**

Friends

- class **Matrix**
- class **MatrixCell**
- class **ColPair**

8.7.1 Detailed Description

One collidable object.

Holds various flags for one collidable object, including all auxiliary collision detection data (hierarchies, etc.).

Todo

Was man noch tun muss ..

Implementation:

Each ColObj stores the toWorld matrix of the geometry, because the collision detection module does not necessarily run in its own thread, and thus does not necessarily have its own aspect.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 col::ColObj::ColObj (osg::GeometryPtr & *geom*, osg::NodePtr & *node*, bool *flexible*, bool *stationary*, bool *compute_hull*, AlgoE *algo*, Grid * *grid*, bool *show_hull* = `false`, char * *colname* = `NULL`)

The ctor you should use.

Parameters:

geom,node the OSG object

flexible tells the coll.det. module that this object might deform

stationary tells the coll.det. module that this object won't move

compute_hull a convex hull of *m_geom* will be computed and stored,

algo determines what hierarchical data structure to build

show_hull creates a geometry from the convex hull

colname name of colobj

This is the ctor whould should be used for creating ColObj's.

If an object is not m_stationary, then hasMoved() will return true when called for the first time (whether or not the object really has moved).

If *show_hull* is `true`, then the convex hull geometry will be attached to the *m_node* so that it will get rendered, too.

Colname is the name which will be printed with ColObj::print; if it is NULL, then the OSG name will be printed; if that does not exist, an automatically generated ID will be printed.

Precondition:

m_geom should belong to *m_node*.

Exceptions:

XDopTree If geometry has no polygons, or no GeoPosition3f.

XColBug If a bug in the doptree code occurs.

bad_alloc

Todo

Parameter *m_geom* ist ueberfluessig.

See also:

MatrixCell::check

8.7.2.2 col::ColObj::ColObj (const ColObj & *source*)

Copy a collision object.

Warning:

Since the two colobj's point to the same geometry, their DOP trees are copied *only* shallow!

8.7.3 Member Function Documentation

8.7.3.1 void col::ColObj::operator= (const ColObj & *source*)

Assignment.

Warning:

Since the two colobj's point to the same geometry, their DOP trees are copied *only* shallow!

8.7.3.2 void col::ColObj::updateBBox (void)

Update current world bbox.

Calculates the current bbox in world coordinates. It can be retrieved later by getBBox().

Bug

Funktioniert noch nicht, da OSG einen Bug hat.

8.7.3.3 bool col::ColObj::bboxIntersects (ColObj * *other*)

Check if the bboxes of two objects intersect.

Parameters:

other another collision object

8.7.3.4 bool col::ColObj::hasMoved (unsigned int *global_cycle*)

Update toworld matrix and check whether or not an object has moved.

This is based on a toWorld matrix comparison. The check will be performed only once per collision cycle.

Whether or not the obj has moved, *m_curr_matr* will be copied into m_old_matr, and *m_curr_matr* will get the node's current toWorld matrix. This happens only if *global_cycle* has been incremented.

See also:

MatrixCell::check()

8.7.3.5 void col::ColObj::setActive (bool *active*)

Set the activity-status of a Colobj Used to remove an object temporarily from collision checks.

Parameters:

active true => object will be checked

8.7.4 Member Data Documentation

8.7.4.1 int col::ColObj::m_col_matr_idx [protected]

row/column index into col.

interest matrix If *col_matr_idx* < 0, then the ColObj is not valid. However, this should *never* happen!

8.7.4.2 bool col::ColObj::m_has_moved [protected]

Flags whether or not obj has moved since last frame.

Is valid only if *cycle* == *coll::Cycle*

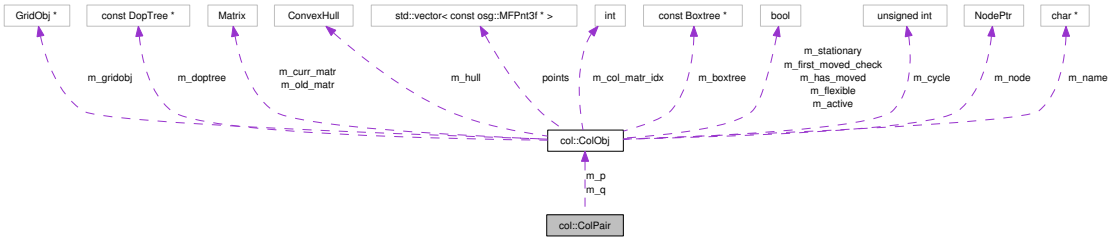
The documentation for this class was generated from the following files:

- ColObj.h
- ColObj.cpp

8.8 col::ColPair Class Reference

Pairs of ColPObjs's.

Collaboration diagram for col::ColPair:



Public Member Functions

- **ColPair** (ColObj *p, ColObj *q)
- **ColPair** (const ColPair &source)
- void **operator=** (const ColPair &source)
- ColObj * p (void) const

  Return one of the two objects of the pair.

- ColObj * q (void) const

  Return the other one of the two objects of the pair.

Protected Attributes

- ColObj * m_p

  the two ColObj's a ColPair consists of

- ColObj * **m_q**

8.8.1 Detailed Description

Pairs of ColPObjs's.

This class is mainly useful for making vectors of colobj pairs.

See also:

ColObj

Todo

Was man noch tun muss ..

Implementation:

Implementierungsdetails, TODO's, etc.

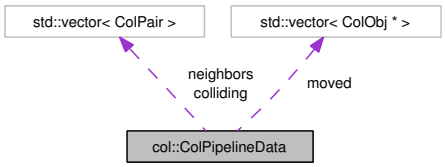
The documentation for this class was generated from the following files:

- ColObj.h
- ColObj.cpp

8.9 col::ColPipelineData Struct Reference

Struct to store some things which are used in Collision Detection Pipeline.

Collaboration diagram for col::ColPipelineData:



Public Member Functions

Constructors, desctructors

- ColPipelineData ()
  Construct a struct with three vectors (moved, neighbors, colliding).

Public Attributes

- vector< ColObj * > moved
  List of objects moved since last frame (in collision pipeline); only growing.
- vector< ColPair > neighbors
  List of neighbors after grid (in collision pipeline); only growing.
- vector< ColPair > colliding
  List of colliding pairs (in collision pipeline); only growing.

8.9.1 Detailed Description

Struct to store some things which are used in Collision Detection Pipeline.

This struct is used in Collision.h and Collision.cpp. It stores three vectors (moved, neighbors, colliding).

Author:

Tobias Ehlgen

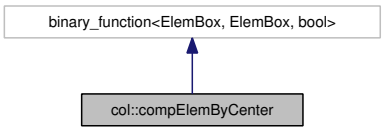
The documentation for this struct was generated from the following files:

- ColPipelineData.h
- ColPipelineData.cpp

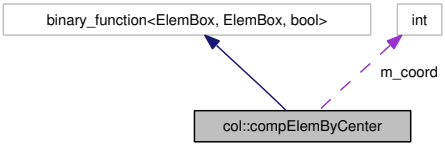
8.10 col::compElemByCenter Struct Reference

Compare two elemntary boxes by the center point along one axis.

Inheritance diagram for col::compElemByCenter:



Collaboration diagram for col::compElemByCenter:



Public Member Functions

- **compElemByCenter** (int coord)
- bool **operator()** (const ElemBox &a, const ElemBox &b)

Public Attributes

- int m_coord

  coord of center that will be compared

8.10.1 Detailed Description

Compare two elemntary boxes by the center point along one axis.

Returns:

True iff self.center[coord] < other.center[coord]

This is a binary predicate for the *sort()* function.

Precondition:

0 <= coord <= 2.

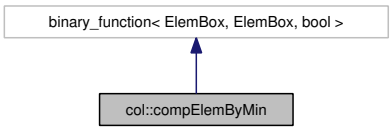
The documentation for this struct was generated from the following file:

- ColBoxtree.cpp

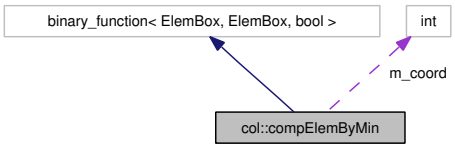
8.11 col::compElemByMin Struct Reference

Compare two elemntary boxes by the center point along one axis.

Inheritance diagram for col::compElemByMin:



Collaboration diagram for col::compElemByMin:



Public Member Functions

- **compElemByMin** (int coord)
- bool **operator()** (const ElemBox &a, const ElemBox &b)

Public Attributes

- int m_coord

  coord of center that will be compared

8.11.1 Detailed Description

Compare two elemntary boxes by the center point along one axis.

Returns:

True iff self.center[coord] < other.center[coord]

This is a binary predicate for the *sort()* function.

Precondition:

0 <= coord <= 2.

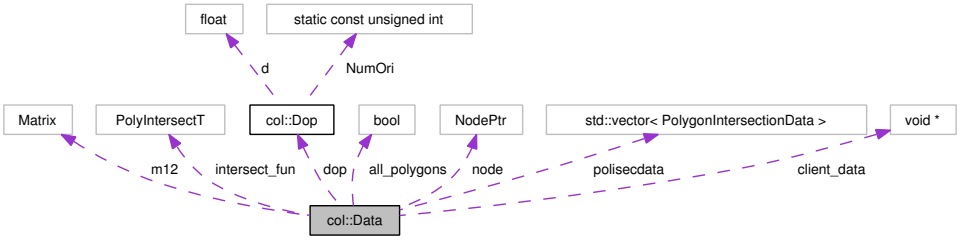
The documentation for this struct was generated from the following file:

- ColBoxtree.cpp

8.12 col::Data Struct Reference

Holds results from collision detection and client data.

Collaboration diagram for col::Data:



Public Member Functions

- Data (const osg::NodePtr &node1, const osg::NodePtr &node2)

  Construct a struct for passing data down to individual polygon checks.

- void addPolygonIntersectionData (const osg::Pnt3f *points1, const osg::Pnt3f *points2, const unsigned int *pgon1, const unsigned int *pgon2, unsigned int nvertices1, unsigned int nvertices2, const osg::GeometryPtr &geom1, const osg::GeometryPtr &geom2, unsigned int pgon_index1, unsigned int pgon_index2)

  Fill polisecdata.

Public Attributes

- std::vector< PolygonIntersectionData > **polisecdata**
- osg::NodePtr node [2]

  Pointers to the two geometries being checked.

- osg::Matrix m12

  Transformation from geom[0] into geom[1]'s frame.

- bool **all_polygons**
- void * client_data

  client data

- PolyIntersectT intersect_fun

  The function for checking a pair of polygons, NULL = built-in.

- const Dop * dop [2]

  Only for debugging; DOPs of leaves in geom[1]'s coord system.

Protected Member Functions

- **Data** (const Data &source)
- Data & **operator=** (const Data &source)

8.12.1 Detailed Description

Holds results from collision detection and client data.

This struct is used to pass data to collision callbacks.

It is also used internally to pass data around within the collision pipeline and recursive collision detection algos.

Todo

- Aus intersect_fun einen Funktor machen.
- Interne Daten vielleicht in eine Unterklasse ziehen.

Author:

Gabriel Zachmann

8.12.2 Constructor & Destructor Documentation

8.12.2.1 col::Data::Data (const osg::NodePtr & *node1*, const osg::NodePtr & *node2*)

Construct a struct for passing data down to individual polygon checks.

Parameters:

node1,node2 the two geometries to be checked for collision

Exceptions:

XCollision If a `geom` does not have a positions array.

Todo

- Ist es ok, die Fkt getPoints() zu verwenden, wenn sich die Punkte waehrend der Koll.erkennung veraendern (durch OSG)?

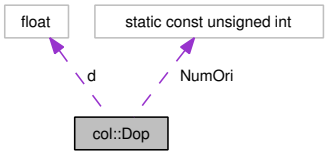
The documentation for this struct was generated from the following files:

- Collision.h
- Collision.cpp

8.13 col::Dop Struct Reference

A DOP is represented by NumOri (=k) many plane offsets.

Collaboration diagram for col::Dop:



Access, Attributes

- float & operator[ ] (const unsigned int k)
  Get k-th component of DOP.
- float operator[ ] (const unsigned int k) const
  Get k-th component of DOP, result is const.
- float max (unsigned int *k=NULL) const
  Get max component of a Dop.
- bool isDegenerate (void) const
  Check whether DOP is degenerate.
- static unsigned int mostParallelOri (const Vec3f &diag, Vec3f *ori=NULL)
  Find the orientation which is "most parallel" to a given vector.

Public Member Functions

Constructors

- Dop ()
  Default ctor.
- Dop (const Dop &source)
  Copy constructor.
- Dop (const Pnt3f &pnt)
  Initialize DOP from a single point.
- Dop (const Dop *source)
  Construct from pointer.
- void setValues (float val[NumOri])
  Set coefficients of DOP explicitely.

Operators

- void operator+= (const Dop &other)
  Increase DOP by extent of other DOP (if necessary).
- void operator+= (const Pnt3f &pnt)
  Increase DOP to include point.
- void operator+= (float delta)
  Increase DOP by a delta.
- void operator-= (const Dop &other)
  Subtract DOP components.
- Dop operator * (const DopTransform &tf) const
  Transform a DOP.
- void extend (float delta)
  Increase DOP by a delta.

Assignment, Initialization

- void operator= (const Pnt3f &pnt)
  Initialize DOP from a single point.
- void operator= (const Dop &other)
  Initialize DOP from another DOP.
- void operator= (float f)
  Initialize all components of the DOP with the same value.

Comparison

- bool operator== (const Dop &other) const
  Check if all coefficients are (almost) the same.
- bool operator!= (const Dop &other) const
  Check if some coefficient is unequal.
- bool overlap (const Dop &other) const
  Check whether two DOPs overlap.

Debugging

- void print (void) const
  Print all components of a Dop.
- osg::NodePtr getGeom (void) const
  Create a polyhedron from a DOP.

Public Attributes

- float **d** [NumOri]

Static Public Attributes

- static const unsigned int NumOri = 24

  number of orientations of Dop's (= k)

8.13.1 Detailed Description

A DOP is represented by NumOri (=k) many plane offsets.

Author:

Gabriel Zachmann

Todo

8.13.2 Constructor & Destructor Documentation

8.13.2.1 col::Dop::Dop ()

Default ctor.

Components are *not* initialized!

8.13.2.2 col::Dop::Dop (const Pnt3f & *pnt*)

Initialize DOP from a single point.

Parameters:

pnt The point

Each component of the DOP is the projection of `pnt` onto the corresponding orientation.

Todo

Use OSG's dotprod(vec,pnt) when available.

8.13.2.3 col::Dop::Dop (const Dop * *source*)

Construct from pointer.

Parameters:

source another DOP

If source==NULL, all components will be *uninitialized* (like default), if source!=NULL, all components will be copied from source. Needed for DopNode::check().

8.13.3 Member Function Documentation

8.13.3.1 void col::Dop::setValues (float *val*[NumOri])

Set coefficients of DOP explicitely.

For debugging.

Exceptions:

XDopTree If the DOP would be degenerate (i.e., coefficients of anti-parallel orientations are reversed).

Parameters:

val array of values

8.13.3.2 void col::Dop::operator+= (const Dop & *other*)

Increase DOP by extent of other DOP (if necessary).

Parameters:

other another DOP

Warning:

The (left) Dop must have been initialized by Dop::Dop(const Dop &source) or by Dop::operator= .

8.13.3.3 void col::Dop::operator+= (const Pnt3f & *pnt*)

Increase DOP to include point.

Parameters:

pnt point

Warning:

The (left) Dop must have been initialized by Dop::Dop(const Dop &source) or by Dop::operator= .

8.13.3.4 void col::Dop::operator+= (float *delta*)

Increase DOP by a delta.

Warning:

The (left) Dop must have been initialized by Dop::Dop(const Dop &source) or by Dop::operator= .

8.13.3.5 void col::Dop::operator-= (const Dop & *other*)

Subtract DOP components.

Parameters:

other another DOP

8.13.3.6 void col::Dop::operator= (const Pnt3f & *pnt*)

Initialize DOP from a single point.

See also:

Dop::Dop(const Pnt3f &pnt)

8.13.3.7 Dop col::Dop::operator * (const DopTransform & *tf*) const

Transform a DOP.

Parameters:

tf the transformation

See also:

DopTransform::operator*

8.13.3.8 bool col::Dop::operator== (const Dop & *other*) const

Check if all coefficients are (almost) the same.

Parameters:

other DOP

Returns:

True/false

Implementation:

Uses NearZero as epsilon.

8.13.3.9 bool col::Dop::operator!= (const Dop & *other*) const

Check if some coefficient is unequal.

Parameters:

other DOP

Returns:

False, if all coefficient are (almost) equal, true otherwise.

See also:

operator == .

8.13.3.10 float col::Dop::max (unsigned int * *k* = NULL) const

Get max component of a Dop.

Parameters:

k Index of max component (NULL = don't care)

Returns:

The maximum.

8.13.3.11 bool col::Dop::overlap (const Dop & *other*) const

Check whether two DOPs overlap.

Parameters:

other DOP

Returns:

True if the 2 DOPs overlap, false otherwise.

Implementation:

= ovrlpBVs in Y.

8.13.3.12 bool col::Dop::isDegenerate (void) const

Check whether DOP is degenerate.

Returns:

True if it is degenerate, i.e., (d[k] < -d[NumOri/2+k]) or (-d[k] > d[NumOri/2+k]) for some k.

8.13.3.13 void col::Dop::extend (float *delta*)

Increase DOP by a delta.

Warning:

The (left) Dop must have been initialized by Dop::Dop(const Dop &source) or by Dop::operator= .

8.13.3.14 unsigned int col::Dop::mostParallelOri (const Vec3f & *diag*, Vec3f * *ori* = NULL) [static]

Find the orientation which is "most parallel" to a given vector.

Parameters:

diag Vector

ori The orientation found (out; NULL=ok)

Returns:

Index of the "most parallel" orientation (which is `ori`).

The "most parallel" orientation is the one with the largest dot product with diag.

8.13.3.15 osg::NodePtr col::Dop::getGeom (void) const

Create a polyhedron from a DOP.

Returns:

The node.

Create a polyhedron which is the convex object described by the planes given by DopTree::m_Ori[], and d[]. The geometry will have no material.

Every time this method is called, a *new* geometry is created, even if it is the same Dop!

All Dop's share the same material.

Exceptions:

XColBug If there are too many points (*NumPnts*). (shouldn't happen)

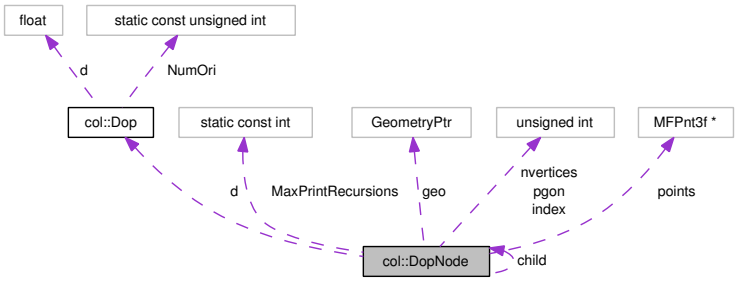
The documentation for this struct was generated from the following files:

- ColDopTree.h
- ColDopTree.cpp

8.14 col::DopNode Struct Reference

DOP node of the DOP hierarchy.

Collaboration diagram for col::DopNode:



Debugging

- void print (int depth, bool print_dops) const

  Print DOP tree.

- unsigned int numFaces (void) const

  Return number of faces under a DOP node.

- osg::NodePtr getGeom (int level) const

  Create geometry from DOP tree.

- void **getGeom** (int level, osg::NodePtr &root) const

Public Member Functions

- bool **check** (DopNode &other, Data *data)

Debugging

- DopNode ()

  Init DOP tree node.

- bool check_down (const DopNodeList &other, Data *data, const DopTransform &dt) const

  Recursive work-horse for DopTree::check().

- bool check_stay (const DopNodeList &other, Data *data, const Dop &e, const DopTransform &dt) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. Like previous method, but with the additional condition that this *is a leaf node, so we actually stay on the same node as of the previous recursion.*

Public Attributes

- Dop d
  DOP for this node.
- DopNode * child [2]
  Child contains pointer to children, or, if node is a leaf, it contains an index to a polygon.
- const osg::MFPnt3f * points
  The enclosed polygon; if leaf node = indices into vertex array.
- unsigned int **pgon** [3]
- unsigned int **nvertices**
- osg::GeometryPtr geo
  OSG index of enclosed polygon.
- unsigned int **index**

Static Protected Attributes

- static const int **MaxPrintRecursions** = 80

8.14.1 Detailed Description

DOP node of the DOP hierarchy.

Author:

Gabriel Zachmann

Todo

Kann nur Dreiecke handeln! (siehe *nvertices* im header file!)

8.14.2 Constructor & Destructor Documentation

8.14.2.1 col::DopNode::DopNode ()

Init DOP tree node.

DOP components are set to

```
-numeric_limits<float>::max()
```

.

8.14.3 Member Function Documentation

8.14.3.1 bool col::DopNode::check_down (const DopNodeList & *other*, Data * *data*, const DopTransform & *dt*) const

Recursive work-horse for DopTree::check().

Parameters:

other the other DopTree (bvh2)

dt DOP transformation from this coords into other's coords

data recursion data (e.g., DopTransform), and callback data, e.g., intersecting polygon(s)

Returns:

True, if intersection, false otherwise.

This is the traversal scheme from "High-Performance Collision Detection Hardware" Technical Report, Aug 2003, University Bonn, see http://web.informatik.uni-bonn.de/II/ag-klein/people/zach/papers/collchip_techrep.html This method actually processes *this*->*child*[0] and this->child[1]!

If data->intersect_fun (PolyIntersectT) is set, this will be called for an intersection test of pairs of polygons.

Precondition:

- the children of this (child[0] and child[1]) are tested
- The two geometries are already stored in `data`.
- The orientations in `Ori` *must* be pairwise anti-parallel!
- A node is an inner node iff child[0] is NULL.

Todo

- const machen
- Check, ob pairwise processing hier in SW etwas bringt
- DopNode sollte 2 Dop's enthalten (wie der Algo es eben braucht)
- mit Intel compiler nochmal checken, ob transform2 nicht doch etwas bringt. Dito mit overlap2.
- Verbesserungen aus VRST-Paper einbauen
- In den arrays *child_other* koennte man den 2ten Zeiger einsparen, da immer other[i] und other[i]+1 betrachtet werden muessen. Dann waeren die Listen nur halb so gross.

Implementation:

self = bvh1, other = bvh2, dt = comn->Bb/d/o, ee = e1, compared to Y:opyIntersectR.

8.14.3.2 bool col::DopNode::check_stay (const DopNodeList & *other*, Data * *data*, const Dop & *e*, const DopTransform & *dt*) const

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. Like previous method, but with the additional condition that *this* is a leaf node, so we actually stay on the same node as of the previous recursion.

Todo

: Per overload deklarieren.

8.14.3.3 osg::NodePtr col::DopNode::getGeom (int *level*) const

Create geometry from DOP tree.

Parameters:

level a polyhedron will be created for all nodes on this level

If a leaf has a level less than *level*, then a geometry will still be created for it. If `level` < 0, then only leaves will be considered.

See also:

Dop::getGeom

8.14.4 Member Data Documentation

8.14.4.1 DopNode* col::DopNode::child[2]

Child contains pointer to children, or, if node is a leaf, it contains an index to a polygon.

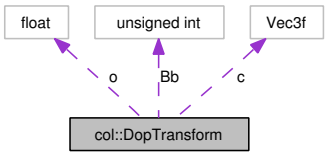
The documentation for this struct was generated from the following files:

- ColDopTree.h
- ColDopTree.cpp

8.15 col::DopTransform Struct Reference

Affine transformation for DOP re-alignment.

Collaboration diagram for col::DopTransform:



Public Member Functions

General functions

- DopTransform (const osg::Matrix &m)

  Calculate DOP transformation from a matrix.

- DopTransform (const osg::Quaternion &q)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- void operator= (const osg::Matrix &m)

  Calculate DOP transformation from a matrix.

- void operator= (const osg::Quaternion &q)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- void print (void) const

  Print a DOP transformation.

- Dop operator * (const Dop &dop) const

  Transform a DOP.

Public Attributes

- Vec3f c [Dop::NumOri]

  The transformation matrix.

- float o [Dop::NumOri]

  The translation of the affine transformation.

- unsigned int Bb [Dop::NumOri][3]

  The correspondence between old and new DOP coefficients.

8.15.1 Detailed Description

Affine transformation for DOP re-alignment.

Author:

Gabriel Zachmann

8.15.2 Constructor & Destructor Documentation

8.15.2.1 col::DopTransform::DopTransform (const osg::Matrix & *m*)

Calculate DOP transformation from a matrix.

Parameters:

m the transformation matrix (rot + transl)

The result can be used for operator∗ .

Sideeffects:

Class variables m_Ori, m_Vtx2Ori, m_Pnt will be used.

Implementation:

= opyCorrespFromMat in Y.

8.15.3 Member Function Documentation

8.15.3.1 void col::DopTransform::operator= (const osg::Matrix & *m*)

Calculate DOP transformation from a matrix.

See also:

DopTransform::DopTransform(const osg::Matrix &m)

8.15.3.2 Dop col::DopTransform::operator ∗ (const Dop & *dop*) const

Transform a DOP.

Parameters:

dop a Dop

If the transformation has been computed with DopTransform(osg::Matrix&), then the result will be an axis-aligned DOP enclosing the original one.

Implementation:

= opyCalcPlaneOffsets in Y. It seems that the double indexing costs about half the time!

The documentation for this struct was generated from the following files:

- ColDopTree.h
- ColDopTree.cpp

8.16 DopTree Class Reference

DOP-tree collision check algorithm.

8.16.1 Detailed Description

DOP-tree collision check algorithm.

For an extensive explanation of the algorithms, please see my dissertation at http://www.gabrielzachmann.org/ .

The construction of a DopTree uses the FaceIterator to retrieve a list of all polygons of an OSG geometry.

Author:

Gabriel Zachmann, written May 1997, ported to OSG Jan 2001.

Compile-Time Flags:

- `LEAFHANDLER` - if set, each DOPtree can have it's own function for handling leaves; if not set, leaves will always be handled by intersectpoly. Setting this will cost a little memory and performance, but is useful for testing.
- `SECOND_ITERATION_FOR_DIAMETER` - does another iteration in order to find the pair of elementary DOPs furthes apart from each other.
- `KRITERIUM` - defines the algorithm for splitting a set of elementary DOPs.
- `DOPTREE_NUM_ORI` - number of orientations; always = class variable NumOri; needed for ifdef's in init().

Todo

- Creation of DOP trees is *very* slow!
- Resolve all the TODO's in the code.
- Check if SECOND_ITERATION_FOR_DIAMETER really helps.
- Keine virtual methods (dtor)! (wg. Speicherplatz fuer vtable)
- DOP tree in 1 grosses Array speichern!
- Does performance increase, if all local variables are doubles?
- Zwei verschiedene DopNode's? eine Klasse fuer innere Knoten, eine fuer Blaetter; wg. Speicher fuer pgon, der in inneren Knoten nicht gebraucht wird! Evtl. kann man das auch durch union { d; pgon } machen.
- Instanzvariable *index* braucht man nicht, da das in *child*[1] gespeichert werden kann; Instanzvariable *nvertices* braucht man nicht, wenn man festlegt, dass sowieso nur 3- oder 4-ecke gespeichert werden koennen, und man pgon[3] fuer 3-ecke einfach auf *MAX_UINT* setzt.
- Klasse DopTree in Klasse DopNode mergen (oder umgekehrt).
- Verschiedene Farben bei DOP-Tree-Visualisierung.
- Code den Naming-Konventionen anpassen! (besonders m_var und M_Var)

Implementation:

The affine transformation for re-aligning DOPs is a separate struct, and not a bunch of instance variables of DopTree, so that the intersection test function can be made thread-safe.

Implementation:

See also vd2/y/oripoly.c for the original version of this code. Some of that has been omitted here, because it was experimental and has not proven to improve performance.

Bug

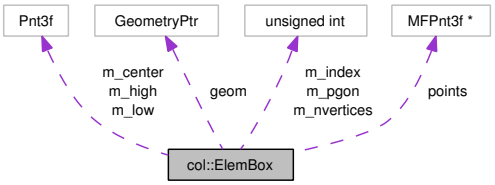
The documentation for this class was generated from the following file:

- ColDopTree.cpp

8.17 col::ElemBox Class Reference

Elementary box, enclosing one polygon, for Boxtree.

Collaboration diagram for col::ElemBox:



Public Member Functions

- ElemBox ()

  Default constructor.

- ElemBox (const osg::FaceIterator &fi, const osg::MFPnt3f *points)

  Constructor.

- void set (const osg::FaceIterator &fi, const osg::MFPnt3f *points)

  Copy a polygon into an elementary box.

- void operator= (const ElemBox &source)

  Copy elementary box.

- bool operator== (const ElemBox &other) const

  Two elementary boxes are equal, if their polygons are equal.

Static Public Member Functions

- static void calcBox (const vector< ElemBox > &elem, const int left, const int right, Pnt3f *low, Pnt3f *high)

  Compute bbox enclosing all elementary boxes.

Public Attributes

- Pnt3f **m_low**
- Pnt3f **m_high**
- const osg::MFPnt3f * **points**
- unsigned int **m_pgon** [Boxtree::M_MaxNVertices]
- unsigned int **m_nvertices**
- osg::GeometryPtr **geom**
- unsigned int **m_index**
- Pnt3f **m_center**

8.17.1 Detailed Description

Elementary box, enclosing one polygon, for Boxtree.

This is a helper class for Boxtree::build() only!

Author:

Gabriel Zachmann, written 1997, re-implemented on OSG in Mar 2002.

8.17.2 Constructor & Destructor Documentation

8.17.2.1 col::ElemBox::ElemBox (const osg::FaceIterator & *fi*, const osg::MFPnt3f * *nodepoints*)

Constructor.

See also:

ElemBox::set

8.17.3 Member Function Documentation

8.17.3.1 void col::ElemBox::set (const osg::FaceIterator & *fi*, const osg::MFPnt3f * *nodepoints*)

Copy a polygon into an elementary box.

Parameters:

fi face iterator

nodepoints the array of points of the geometry

Copies the number of vertices, vertex indices, and pgon's index of the polygon the FaceIterator *fi* is pointing to into the elementary box. The index is the one returned from the iterator, *not* the one in the Geometry's array!

Calculate and set the barycenter of the elementary box.

Todo

Warum werden bei GL_QUAD_STRIP die letzten beiden Vertices *immer* geswappt??

8.17.3.2 bool col::ElemBox::operator== (const ElemBox & *other*) const

Two elementary boxes are equal, if their polygons are equal.

Bug

Does not work if the polygons are the same but the start index is different!

8.17.3.3 void col::ElemBox::calcBox (const vector< ElemBox > & *elem*, const int *left*, const int *right*, Pnt3f * *low*, Pnt3f * *high*) [static]

Compute bbox enclosing all elementary boxes.

Parameters:

elem array of elementary boxes

left,right consider only elems from left..right (inclusively)

low,high resulting bbox (out)

The box will not be increased by *NearZero*, because this has already been done for the elementary boxes.

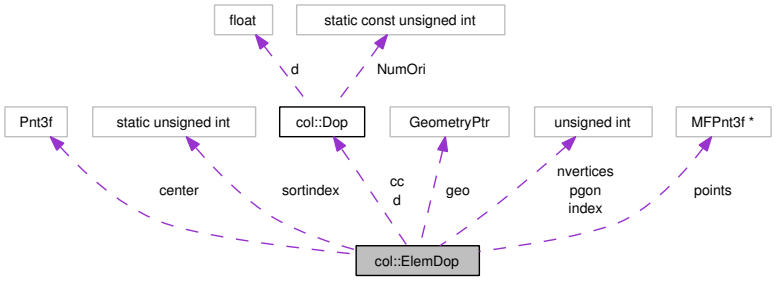
The documentation for this class was generated from the following files:

- ColBoxtree.h
- ColBoxtree.cpp

8.18 col::ElemDop Struct Reference

Elementary DOP enclosing one polygon.

Collaboration diagram for col::ElemDop:



Debugging

- static unsigned int sortindex
- bool operator< (const ElemDop &other) const

  Comparison operators.

- bool **operator>** (const ElemDop &other) const
- bool **operator<=** (const ElemDop &other) const
- bool **operator>=** (const ElemDop &other) const
- void **operator=** (const ElemDop &other)
- static void **setSortIndex** (unsigned int index)

Public Attributes

- Dop **d**
- const osg::MFPnt3f * **points**
- unsigned int **pgon** [MaxNVertices]
- unsigned int **nvertices**
- osg::GeometryPtr **geo**
- unsigned int **index**
- Pnt3f **center**
- Dop **cc**

8.18.1 Detailed Description

Elementary DOP enclosing one polygon.

This is only needed for the construction process

Author:

Gabriel Zachmann

Warning:

Sorting an array/vector of ElemDop's is *not* thread-safe, because the sortindex is a class variable!

Todo

8.18.2 Member Function Documentation

8.18.2.1 bool col::ElemDop::operator< (const ElemDop & *other*) const

Comparison operators.

Returns:

Text for return value.

Needed for sorting a vector of ElemDop's.

The DOPs should have been blown up a little already; no epsilon threshold is included in the comparison.

Todo

Sort mit ordentlichem BinaryPredicate machen! (dann ist das auch thread-safe)

See also:

DopTree::constructHier

8.18.3 Member Data Documentation

8.18.3.1 unsigned int col::ElemDop::sortindex [static]

Todo

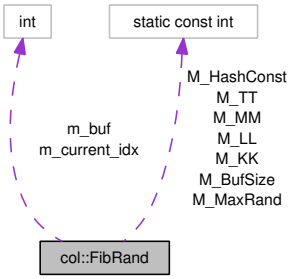
The documentation for this struct was generated from the following files:

- ColDopTree.h
- ColDopTree.cpp

8.19 col::FibRand Class Reference

Lagged Fibonacci random sequence.

Collaboration diagram for col::FibRand:



Random numbers

- **FibRand** (int seed)
- unsigned int rand (void)

  Returns a pseudo random integer in the range [0,FibRand::M_MaxRand).

- unsigned int mrand (unsigned int m)

  Returns a pseudo random integer in the range [0,m).

- float frand (void)

  Returns a pseudo random float in the range [0,1).

Static Public Attributes

- static const int M_MaxRand = (1 << 30)

  Maximum random number plus 1 that can be returned by FibRand.

8.19.1 Detailed Description

Lagged Fibonacci random sequence.

The constructor creates an object, from which you can retrieve random numbers by any of the functions FibRand::rand(), FibRand::mrand(), and FibRand::frand(). fills a field of 100 random numbers, which can be retrieved

Bug

The range has not really been checked/verified.

Implementation:

Nach Knuth TAOCP Vol.2 pp.186f. Internally, a FibRand object stores 100 random numbers, which are then doled out by one of the "getters". After that, a new set of 100 numbers is generated. Hoefully, this yields better performance than producing each random number on demand.

8.19.2 Member Function Documentation

8.19.2.1 unsigned int col::FibRand::rand (void)

Returns a pseudo random integer in the range [0,FibRand::M_MaxRand).

See also:

FibRand

8.19.2.2 unsigned int col::FibRand::mrand (unsigned int *m*)

Returns a pseudo random integer in the range [0,m).

See also:

FibRand

8.19.2.3 float col::FibRand::frand (void)

Returns a pseudo random float in the range [0,1).

See also:

FibRand

The documentation for this class was generated from the following files:

- ColUtils.h
- ColUtils.cpp

8.20 Grid Class Reference

Grid for collision detection.

8.20.1 Detailed Description

Grid for collision detection.

The grid is intended to speed up collision detection by reducing the number of exact collision tests with objects which share some GridCell s.

See also:

GridCell, GridObj

Bug

The *Grid* class and friends are probably not multi-thread-safe, i.e., several threads asking the same grid for a list of intersecting boxes will get different (wrong) answers. (This is because of the cycle counters.)

The documentation for this class was generated from the following file:

- ColGrid.cpp

8.21 GridCell Class Reference

Cells of the grid.

8.21.1 Detailed Description

Cells of the grid.

These objects work as the cells of the grid. They store which objects are (partly) inside of them

See also:

Grid, GridObj

The documentation for this class was generated from the following file:

- ColGridCell.cpp

8.22 GridObj Class Reference

Objects in a grid.

8.22.1 Detailed Description

Objects in a grid.

Representation of scenegraph objects inside the grid, in GridCells.

See also:

Grid, GridCell

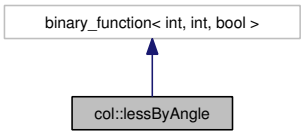
The documentation for this class was generated from the following file:

- ColGridObj.cpp

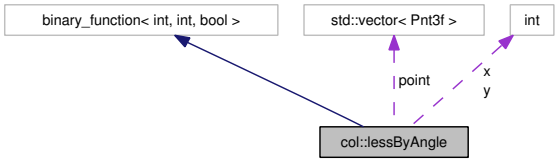
8.23 col::lessByAngle Struct Reference

Compare points by angle.

Inheritance diagram for col::lessByAngle:



Collaboration diagram for col::lessByAngle:



Public Member Functions

- **lessByAngle** (int xValue, int yValue, vector< Pnt3f > &pointVector)
- bool **operator()** (int i1, int i2) const

Public Attributes

- int x

  points will be projected onto that plane

- int **y**
- vector< Pnt3f > & point

  the points themselves; the () expects to get 2 indices into this array

8.23.1 Detailed Description

Compare points by angle.

Returns:

True if point[i1] < point[i2] compared by angle, flse otherwise.

Two points are connected with a "center" by a line each, then the angle between those two lines and the `x` axis are computed, and those are compared. This functor is meant as a Binary Predicate for `sort()`.

Angles are in [0,360). No trigonometric function are evaluated, and if the two points are in different quadrants, then only comparisons are made, i.e. / quadrant(p1) < quadrant(p2), p1,p2 in different qu's p1 < p2 <=> | \ p1_y/p1_x < p2_y/p2_x , p1,p2 in same quadrant

Precondition:

`x`, `y`, and `point`,npoints are set.

Implementation:

The functor could return -1/0/+1 for less-than/euqal/greater-than. The original of this function is opyCompPointsByAngle in Y.

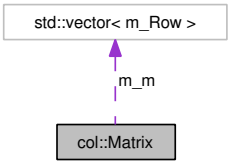
The documentation for this struct was generated from the following file:

- ColUtils.cpp

8.24 col::Matrix Class Reference

The collision interest matrix.

Collaboration diagram for col::Matrix:



Public Member Functions

- Matrix (unsigned int numcolobjs=50)
  Create a collision interest matrix.
- void addObj (ColObj *obj)
  Make a new row & column for a collision object.
- void addCallback (Callback *callback, vector< ColObj > *colobjs)
  Add a callback to a cell of the matrix.
- MatrixCell * getCell (const ColPair &pair) const
  Return the cell corresponding to a colobj pair.
- MatrixCell * createCell (const ColObj *obj1, const ColObj *obj2)
  Create a new collision interest matrix cell.
- bool check (const ColPair &pair, bool use_hulls, AlgoE algo) const
  Check a pair for collision.
- void callCallbacks (const ColPair &pair) const
  Call all callbacks associated with a certain pair of col.
- bool isConsistent (vector< ColObj > *colobjs) const
  Check consistency of the collision interest matrix.

Protected Types

- typedef vector< MatrixCell * > m_Row
  one row of the collision interest matrix (rows have different length)

Protected Member Functions

- **Matrix** (const Matrix &source)
- Matrix & **operator=** (const Matrix &source)

Protected Attributes

- vector< m_Row > m_m
  the raison d'etre

8.24.1 Detailed Description

The collision interest matrix.

Composed of MatrixCell's.

Author:

Gabriel Zachmann

8.24.2 Constructor & Destructor Documentation

8.24.2.1 col::Matrix::Matrix (unsigned int *numcolobjs* = 50)

Create a collision interest matrix.

Parameters:

numcolobjs this is just an estimate of how many objects there will be

The number of collision objects can be incremented later.

Implementation:

Only the lower triangle of the matrix is occupied, i.e., only cells (i,j) are valid with i>j.

8.24.3 Member Function Documentation

8.24.3.1 void col::Matrix::addObj (ColObj * *obj*)

Make a new row & column for a collision object.

Parameters:

obj the collision object

A row is added to the matrix.

Exceptions:

XColBug If *colobj* has already been added.

Warning:

Dinge, die der Aufrufer unbedingt beachten muss...

Precondition:

Annahmnen, die die Funktion macht...

Sideeffects:

Nebenwirkungen, globale Variablen, die veraendert werden, ..

Todo

Was noch getan werden muss

Bug

Bekannte Bugs dieser Funktion

See also:

...

Implementation:

Implementierungsdetails, TODOs, ...

8.24.3.2 void col::Matrix::addCallback (Callback * *callback*, vector< ColObj > * *colobjs*)

Add a callback to a cell of the matrix.

Parameters:

callback the callback

Exceptions:

XCollision If the same *m_callback*, or a callback with the same objects, has been added already.

XCollision If the nodes have not been made collidable yet (by an ADD_OBJECT request).

XColBug

Warning:

A pointer to the callback-functor is stored, so the application should not delete the object.

Precondition:

Annahmnen, die die Funktion macht...

Todo

Implementation:

We cannot check earlier whether or not the nodes have been made collidable, because that could have happened by just one request earlier in the same queue during the same collision cycle.

8.24.3.3 MatrixCell * col::Matrix::getCell (const ColPair & *pair*) const

Return the cell corresponding to a colobj pair.

Parameters:

pair the pair of collision objects

Returns:

The corresponding cell.

Exceptions:

XColBug If either of the colmatrix indices is < 0.

Precondition:

- Matrix index of both *obj must* be valid.
- obj1 != obj2, and index1 != index2.

8.24.3.4 MatrixCell * col::Matrix::createCell (const ColObj * *obj1*, const ColObj * *obj2*)

Create a new collision interest matrix cell.

Parameters:

obj1,obj2 the pair of collision objects

Returns:

The corresponding cell.

Exceptions:

XColBug If either of the colmatrix indices is < 0.

Precondition:

- Matrix index of both *obj must* be valid.
- obj1 != obj2, and index1 != index2.

8.24.3.5 bool col::Matrix::check (const ColPair & *pair*, bool *use_hulls*, AlgoE *algo*) const

Check a pair for collision.

See also:

MatrixCell::check()

8.24.3.6 void col::Matrix::callCallbacks (const ColPair & *pair*) const

Call all callbacks associated with a certain pair of col.

objects

Parameters:

pair the pair of col. objects

If there are no callbacks associated with this pair, nothing happens.

Warning:

Dinge, die der Aufrufer unbedingt beachten muss...

Precondition:

- Matrix index of both *obj must* be valid.
- obj1 != obj2, and index1 != index2.

Sideeffects:

Nebenwirkungen, globale Variablen, die veraendert werden, ..

Todo

Was noch getan werden muss

Bug

Bekannte Bugs dieser Funktion

See also:

...

Implementation:

Implementierungsdetails, TODOs, ...

8.24.3.7 bool col::Matrix::isConsistent (vector< ColObj > * *colobjs*) const

Check consistency of the collision interest matrix.

Parameters:

colobjs the list of collision objects

Returns:

True, if an inconsistency has been detected.

If an inconsistency is detected, an error message is printed. Checks:

- strict lower triangle;

- *colobjs* must fit to the matrix

Precondition:

Implementation:

Implementierungsdetails, TODOs, ...

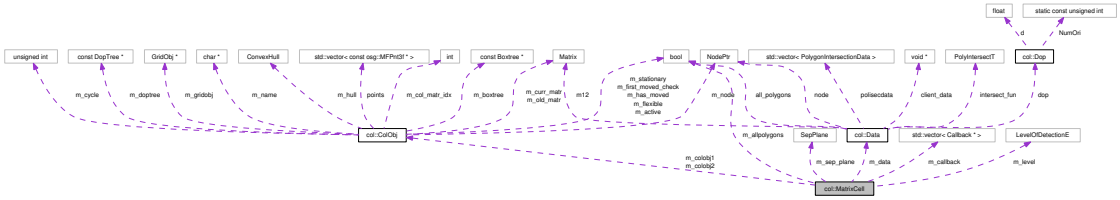
The documentation for this class was generated from the following files:

- ColObj.h
- ColObj.cpp

8.25 col::MatrixCell Class Reference

A single cell of the collision interest matrix.

Collaboration diagram for col::MatrixCell:



Public Member Functions

- **MatrixCell** (const ColObj ∗colobj1, const ColObj ∗colobj2)
- void addCallback (Callback ∗callback)

  Add a collision callback.

- void callCallbacks (void) const

  Process all callbacks.

- bool check (bool use_hulls)

  Check a pair for collision (internal).

Protected Attributes

- vector< Callback ∗ > m_callback

  positive collision callbacks

- ColObj const ∗const m_colobj1

  the two collision objects of this cell

- ColObj const ∗const **m_colobj2**
- LevelOfDetectionE m_level

  the maximum level of detection of all callbacks of this cell

- SepPlane m_sep_plane

  the seprating plane of the convex hulls ColObj::hull

- Data m_data

  Collision data for collision callback and internal usage.

- bool **m_allpolygons**

8.25.1 Detailed Description

A single cell of the collision interest matrix.

Each cell contains a list of Callback's, and other pairwise data (like separating plane).

Each cell also contains a "level of collision". The minimum level is *LEVEL_BOX*. When a cell checks the pair of objects for collision, the maximum level of all callbacks is used for that check.

Exceptions:

XCollision If one of the nodes does not have a geometry.

Todo

Flag *all_poygons* auswerten.

Author:

Gabriel Zachmann

Implementation:

When different algorithms will be available, a cell will be the place to store the kind of algo appropriate for a certain pair of objects.

8.25.2 Member Function Documentation

8.25.2.1 void col::MatrixCell::addCallback (Callback * *cb*)

Add a collision callback.

Exceptions:

XColBug If m_callback->m_node1/2 doesn't match cell.m_colobj1/2->m_node.

XCollision If one of the objects pointers in the callback is a NullNode.

8.25.2.2 void col::MatrixCell::callCallbacks (void) const

Process all callbacks.

Precondition:

m_data is valid.

8.25.2.3 bool col::MatrixCell::check (bool *use_hulls*)

Check a pair for collision (internal).

Parameters:

use_hulls do a convex hull pre-check

Depending on the levels of detection of each callback, the max level needed is done. For instance, if all callbacks have level LEVEL_HULL or less, then only the convex hull check is done.

Implementation:

The check is based on the positions of the objects stored in ColObj::m_curr_matr .

Warning:

Only one of the instance variables `m_doptree` and `m_boxtree` should be set! It will call the check() function of the one which is set. And both ColObj's in a cell should have the same instance variables set, and the other unset!

See also:

ColObj::hasMoved()

Todo

- Check whether or not only a bbox check is wanted. This would be a flag stored with each Callback, and a counter stored with the MatrixCell.
- Matrix-Inversion in ColObj::hasMoved() machen.
- Nochmal ueberpruefen, warum die berechnete Matrix m12 so stimmt; eigtl. haette ich jetzt doch eine umgekehrte Multiplikation erwartet.
- *use_hulls* in jeder MatrixCell speichern. Dann braucht man nicht das Flag global fuer alle MatrixCell's in Collision.cpp sich zu merken.

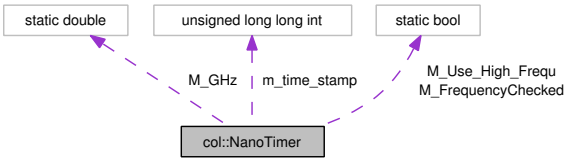
The documentation for this class was generated from the following files:

- ColObj.h
- ColObj.cpp

8.26 col::NanoTimer Class Reference

Timer with nanoseconds resolution.

Collaboration diagram for col::NanoTimer:



Timers, timing, sleeping, etc.

- NanoTimer ()

  Create a new timer with nanoseconds resolution.

- void start (void)

  Save the current time (stamp) in the timer.

- double elapsed (void) const

  Return the time elapsed since the last start() in nanoseconds.

- static bool usesHighFrequ (void)

  Tells whether or not the NanoTimer use the high frequency counter.

- static double frequ (void)

  Returns the frequency (resolution) of the counter in GHz.

8.26.1 Detailed Description

Timer with nanoseconds resolution.

The units of this timer are always nanoseconds, but on some platforms, the actual resolution might be less (microseconds or even less).

Where implemented, this class uses the high-speed, high-performance, hardware timers/counters. Otherwise, we just use *gettimeofday()*, which, at least on Linux/Pentium, has microsecond resolution.

Currently, the time is *wall-clock* time, not user time!

See also:

- http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/IA32LinuxCluster/Doc/timi
- cedar.intel.com/software/idap/media/pdf/rdtscpm1.pdf
- /raphael/knowledge/programming/c/code-snippets/cpu_clock_timer.c

Todo

- Try to estimate the overhead of a function call to start() or elapsed().

- Use PAPI (http://icl.cs.utk.edu/projects/papi/) or the "high resolution timers project" (http://high-res-timers.sourceforge.net/) when they become widely available (without kernel patches).

8.26.2 Constructor & Destructor Documentation

8.26.2.1 col::NanoTimer::NanoTimer (void)

Create a new timer with nanoseconds resolution.

Saves the current time stamp, too.

Sideeffects:

See *checkFrequency()*.

8.26.3 Member Function Documentation

8.26.3.1 bool col::NanoTimer::usesHighFrequ (void) [static]

Tells whether or not the NanoTimer use the high frequency counter.

Warning:

Valid only after the first NanoTimer has been created!

8.26.3.2 double col::NanoTimer::frequ (void) [static]

Returns the frequency (resolution) of the counter in GHz.

Warning:

Valid only if *NanoTimer::usesHighFrequ()* = *true* !

The documentation for this class was generated from the following files:

- ColUtils.h
- ColUtils.cpp

8.27 Request Struct Reference

Collision detection request like "add" or "remove" an object/callback.

8.27.1 Detailed Description

Collision detection request like "add" or "remove" an object/callback.

This struct will be passed to col::request().

Author:

Gabriel Zachmann

Todo

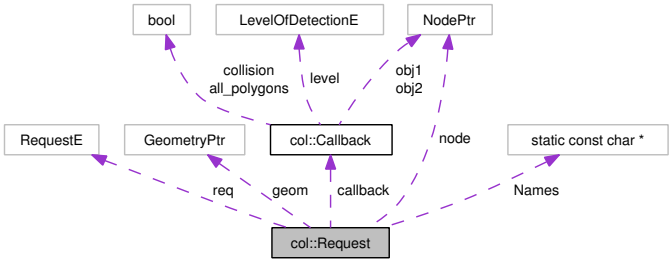
The documentation for this struct was generated from the following file:

- ColRequest.cpp

8.28 col::Request Struct Reference

Each request from the application is encapsulated by an instance of this class.

Collaboration diagram for col::Request:



Public Member Functions

- Request (RequestE req, Callback *callback)

  Create a "two objects request".

- **Request** (RequestE req, osg::NodePtr node)
- void **operator=** (const Request &source)
- void process (bool show_hulls, AlgoE algo, Matrix *colmatrix, std::vector< ColObj > *colobjs, std::vector< Callback * > cycle_callbacks, bool useHulls, Grid *grid)

  Process a request to the collision detection module.

- const char * **getName** (void) const

Public Attributes

- RequestE **req**
- osg::GeometryPtr **geom**
- osg::NodePtr **node**
- Callback * **callback**

Static Public Attributes

- static const char * **Names** [ ]

8.28.1 Detailed Description

Each request from the application is encapsulated by an instance of this class.

In order for the CollisionPipeline to be able to run in parallel to the main ailpcation, requests (such as "register an object") must be queued. This class aides that.

8.28.2 Constructor & Destructor Documentation

8.28.2.1 Request::Request (RequestE *inreq*, Callback * *incallback*)

Create a "two objects request".

Parameters:

inreq the request (ADD_OBJECT,ACTIVATE_OBJECT)

incallback a collision callback

Precondition:

callback should be valid.

Warning:

The constructor does *not* check if the objects in *callback* have already been registered with the collision detection module.

Exceptions:

XCollision If the type of request is not a two object request.

XCollision If *callback* seems to be improperly constructed.

8.28.3 Member Function Documentation

8.28.3.1 void Request::process (bool *show_hulls*, AlgoE *algo*, Matrix * *collmatrix*, std::vector< ColObj > * *colobjs*, std::vector< Callback * > *cycle_callbacks*, bool *useHulls*, Grid * *grid*)

Process a request to the collision detection module.

Warning:

This function probably runs in a different thread than the constructor!

Todo

- Some types not yet implementated.
- process()const machen, wenn OSG erlaubt
- `show_hulls` anders (z.B. als define) implementieren

8.28.4 Member Data Documentation

8.28.4.1 const char * Request::Names [static]

Initial value:

```
{
    "Add object",
    "Add callback",
    "Remove callback",
    "Activate object",
    "Deactivate object",
    "Add cycle callback"
}
```

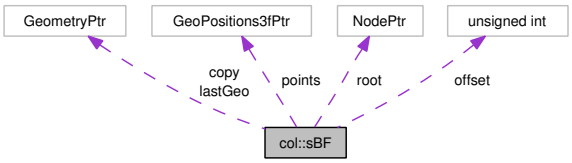
The documentation for this struct was generated from the following files:

- ColRequest.h
- ColRequest.cpp

8.29 col::sBF Struct Reference

contains some state across different invocations of addFace()

Collaboration diagram for col::sBF:



Public Attributes

- unsigned int **offset**
- osg::GeoPositions3fPtr **points**
- osg::GeometryPtr **lastGeo**
- osg::GeometryPtr **copy**
- osg::NodePtr **root**

8.29.1 Detailed Description

contains some state across different invocations of addFace()

The documentation for this struct was generated from the following file:

- ColUtils.cpp

8.30 col::SyncFun Struct Reference

This is a functor for synchronization with other threads.

Public Member Functions

- virtual bool operator() ()=0 throw ()
  This will be executed by the coll.

Collision detection module API

- SyncFun ()
  Create a synchronization functor.

8.30.1 Detailed Description

This is a functor for synchronization with other threads.

Clients of the collision detection module need to derive from this abstract class and overload the () operator.

8.30.2 Member Function Documentation

8.30.2.1 virtual bool col::SyncFun::operator() () throw () [pure virtual]

This will be executed by the coll.

det. module in multithreading mode every time before the check function. If this functor returns 0, then the coll.det. thread (i.e., "coll.det. pipeline") will terminate.

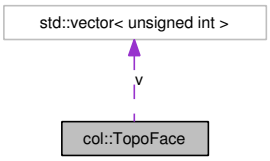
The documentation for this struct was generated from the following files:

- Collision.h
- Collision.cpp

8.31 col::TopoFace Struct Reference

A face is a sorted array of indices into some vertrex array.

Collaboration diagram for col::TopoFace:



Public Member Functions

- TopoFace (const unsigned int vertex_indices[ ], unsigned int size)

  Construct a face from a C array.

- TopoFace (const std::vector< unsigned int > &v)

  Construct a face from a vector.

- TopoFace (const TopoFace &source)

  Copy a face.

- TopoFace (void)

  empty face

- void operator= (const TopoFace &source)

  Copy a face.

- void set (const unsigned int vertex_indices[ ], unsigned int size)

  Copy a face from a C array.

- void print (void) const

  Print the indices of a face (prints no at the end).

- unsigned int & operator[ ] (int index)

  Return vertex index of i-th face vertex.

- unsigned int operator[ ] (int index) const

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- unsigned int size (void) const

  Size of the vertex index array.

- void resize (unsigned int newsize)

  Resize the vertex index array.

Public Attributes

- std::vector< unsigned int > v

  A face is just an array of indices into some vertex array.

8.31.1 Detailed Description

A face is a sorted array of indices into some vertrex array.

Author:

Gabriel Zachmann

8.31.2 Member Function Documentation

8.31.2.1 unsigned int & col::TopoFace::operator[ ] (int *i*)

Return vertex index of i-th face vertex.

Parameters:

i index into face, can be < 0 or >= size()

If the index *i* into the face is out of bounds [0 .. size()-1], then it is wrapped around. So face[-1] returns the same as face[size()-1], for instance.

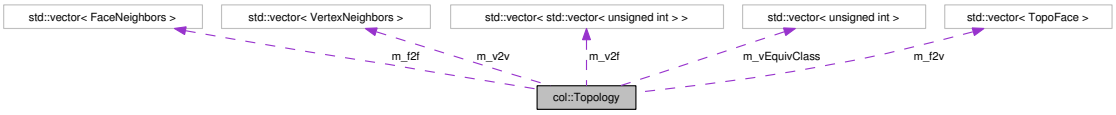
The documentation for this struct was generated from the following files:

- ColTopology.h
- ColTopology.cpp

8.32 col::Topology Class Reference

Zur Beschreibung von Inzidenz- und Adjazenz-Relationen.

Collaboration diagram for col::Topology:



Creation, desctruction, assignments

- Topology ()

  Empty topology.

- **Topology** (const Topology &source)
- void **operator=** (const Topology &source)
- Topology (const std::vector< TopoFace > &face)

  Create the topology relations.

- Topology (const unsigned int face_a[ ][Dop::NumOri], unsigned int nfaces, unsigned int face_nv[ ])

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. Face_a *is a 2-dim.*

- Topology (const osg::GeometryPtr geom, bool unify=true, float tolerance=NearZero)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- void operator= (const std::vector< TopoFace > &face)

  Create the relations from a vector.

- void createFromGeom (const osg::GeometryPtr geom, bool unify=false, float tolerance=NearZero)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. This is exactly like the constructor Topology(const osg::GeometryPtr geom).

- void create (const unsigned int face_a[ ][Dop::NumOri], unsigned int nfaces, unsigned int face_nv[ ])

  Create the relations from arrays.

- void createRelations (void)

  Create relationships.

Public Types

- typedef std::vector< unsigned int > **FaceNeighbors**
- typedef FaceNeighbors::const_iterator **FaceNeighborIterator**
- typedef std::vector< unsigned int > **VertexNeighbors**
- typedef VertexNeighbors::const_iterator **VertexNeighborIterator**

Public Member Functions

- void **setVertexEqualityEpsilon** (OSG::Real32 epsilon)
- void **setVertexEqualityTest** (bool status)

Access

- unsigned int v_neighbors (unsigned int v_index) const
  Return the number of neighbors a vertex has (i.e,, its degree).
- unsigned int v_degree (unsigned int v_index) const
  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- unsigned int v2f_size (unsigned int v_index) const
  Return the number of faces a vertex is incident to.
- unsigned int f_size (unsigned int f_index_in) const
  Return the number of vertices a face has.
- const unsigned int * f_index (unsigned int f_index_in) const
  Return the table of vertex indices for a given face.
- const std::vector< TopoFace > & getFaceVector (void) const
  Return the original face *vector used to construct the topology.*
- Topology::VertexNeighborIterator vertexNeighborBegin (unsigned int v_index) const
  Get begin of vertex neighbors.
- Topology::VertexNeighborIterator vertexNeighborEnd (unsigned int v_index) const
  Get end of vertex neighbors.
- Topology::FaceNeighborIterator faceNeighborBegin (unsigned int f_index) const
  Get begin of face neighbors.
- Topology::FaceNeighborIterator faceNeighborEnd (unsigned int f_index) const
  Get end of face neighbors.
- void print (void) const
  Print the topology (for debugging purposes).

Protected Attributes

- std::vector< std::vector< unsigned int > > m_v2f
  vertex to face incidence relation; see createNeighbors(); v2f[i][j] is an index of a face that contains vertex i .
- std::vector< VertexNeighbors > m_v2v
  vertex to vertex adjacency relation; see createNeighbors(); v2v[i][j] is an index of a vertex that is adjacent to i .
- std::vector< FaceNeighbors > m_f2f

vertex to vertex adjacency relation; see createNeighbors(); f2f[i][j] is an index of a face that is adjacent to i .

- std::vector< TopoFace > m_f2v

  face to vertex incidence relation (the array of faces)

- std::vector< unsigned int > m_vEquivClass

  Equity relation of vertex indices.

8.32.1 Detailed Description

Zur Beschreibung von Inzidenz- und Adjazenz-Relationen.

Alle Relationen verwenden Indices, keine Pointer. Die zugehoerige Geometrie wird (z.Z.) *nicht* in einem Topology-Objekt mit gespeichert.

See also:

Classes ConvexHull, TopoFace.

Todo

- Mehr Iterierungsfunktionen.

Implementation:

Implementierungsdetails, etc.

Author:

Gabriel Zachmann

8.32.2 Constructor & Destructor Documentation

8.32.2.1 col::Topology::Topology ()

Empty topology.

You assign a sensible value from a vector<TopoFace> later with the = operator.

8.32.2.2 col::Topology::Topology (const std::vector< TopoFace > & *face*)

Create the topology relations.

Parameters:

face an array of faces

Each TopoFace of the *face* array consists of a number of indices; each of these indices is an index into some vertex array. So, vertices are just identified by the index. The topology object doesn't need to know the actual vertex array.

Exceptions:

XColBug siehe createRelations

Warning:

The *vertex* and the *face* vectors must *not* change after the Topology has been constructed!

Precondition:

The largest index in *face* determines the number of vertices.

8.32.2.3 col::Topology::Topology (const unsigned int *face_a*[ ][Dop::NumOri], unsigned int *nfaces*, unsigned int *face_nv*[ ])

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. *Face_a* is a 2-dim.

array of indices into some vertex array. *nfaces* is the number of rows in *face_a*; *face_nv*[i] is the number of elements in *face*[i] (which *must* be < *Dop::NumOri!*).

Exceptions:

XCollision If some face_nv[i] > Dop::NumOri.

8.32.2.4 col::Topology::Topology (const osg::GeometryPtr *geom*, bool *unify* = `true`, float *tolerance* = NearZero)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

unify If true (default), then vertices will be unified first

tolerance Threshold for the unification

If the geometry contains GL_POLYGON's, this might cause performance issues, because later, when you walk around a vertex, you need many steps just to "step over" a single polygon. (The constructor will print a warning in this case.)

If *unify* is set, then vertices will first be unified (according to some tolerance), before the topology relations are generated. The geometry *geom* will not be altered.

If the same vertices (= same coordinates) occur with different indices in *geom's* faces, and if *unify* is set to *false*, then they will still *not* be treated as the same vertex, i.e., a walk around such a vertex will *not* find all incident faces!

8.32.3 Member Function Documentation

8.32.3.1 void col::Topology::operator= (const std::vector< TopoFace > & *face*)

Create the relations from a vector.

Exactly like the constructor Topology(const std::vector<TopoFace> &face).

8.32.3.2 void col::Topology::createFromGeom (const osg::GeometryPtr *geom*, bool *unify* = `false`, float *tolerance* = NearZero)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. This is exactly like the constructor Topology(const osg::GeometryPtr geom).

Bug

Es kommen immer 0 faces raus! hat FaceIterator einen Bug?!

See also:

setUnifyTolerance, setUnify

8.32.3.3 void col::Topology::create (const unsigned int *face_a*[ ][Dop::NumOri], unsigned int *nfaces*, unsigned int *face_nv*[ ])

Create the relations from arrays.

Exactly like the constructor Topology(face_a[][Dop::NumOri], ...)

8.32.3.4 unsigned int col::Topology::v_neighbors (unsigned int *m_v_index*) const

Return the number of neighbors a vertex has (i.e,, its degree).

Parameters:

m_v_index the index number of a vertex

Returns:

the number of neighbors a vertex has (i.e,, its degree)

Warning:

m_v_index must be less than the number of vertices!

8.32.3.5 unsigned int col::Topology::v2f_size (unsigned int *m_v_index*) const

Return the number of faces a vertex is incident to.

Parameters:

m_v_index the index number of a vertex

Returns:

the number of faces a vertex is incident to.

Warning:

m_v_index must be less than the number of vertices!

8.32.3.6 unsigned int col::Topology::f_size (unsigned int *f_index_in*) const

Return the number of vertices a face has.

Parameters:

f_index_in the index number of a face

Warning:

f_index must be less than the number of faces!

8.32.3.7 const unsigned int * col::Topology::f_index (unsigned int *f_index_in*) const

Return the table of vertex indices for a given face.

Parameters:

f_index_in the index number of a face

Returns:

the table of vertex indices for a given face

Warning:

f_index_in must be less than the number of faces!

8.32.3.8 Topology::VertexNeighborIterator col::Topology::vertexNeighborBegin (unsigned int *m_v_index*) const

Get begin of vertex neighbors.

Warning:

For performance reasons, we don't check whether or not *m_v_index* is in bounds!

8.32.3.9 Topology::VertexNeighborIterator col::Topology::vertexNeighborEnd (unsigned int *m_v_index*) const

Get end of vertex neighbors.

Warning:

The end iterator points behind the last neighbor! For performance reasons, we don't check whether or not *f_index* is in bounds!

8.32.3.10 Topology::FaceNeighborIterator col::Topology::faceNeighborBegin (unsigned int *inf_index*) const

Get begin of face neighbors.

Warning:

For performance reasons, we don't check whether or not *f_index* is in bounds!

8.32.3.11 Topology::FaceNeighborIterator col::Topology::faceNeighborEnd (unsigned int *inf_index*) const

Get end of face neighbors.

Warning:

The end iterator points behind the last neighbor! For performance reasons, we don't check whether or not *f_index* is in bounds!

8.32.3.12 void col::Topology::createRelations (void) [protected]

Create relationships.

This is meant to be called from the constructor

Exceptions:

XColBug If there is a bug in the vertex or face vectors. And if there is some inconsistency in the topology constructed.

Precondition:

- Instance variable *face* is valid.
- The vertex indices in each *face*[i] are sorted.

Todo

Calc sizes of vectors first, so that we can do a resize() for each vector before filling it, in order to reduce memory fragmentation.

8.32.4 Member Data Documentation

8.32.4.1 std::vector<unsigned int> col::Topology::m_vEquivClass [protected]

Equity relation of vertex indices.

vEequality[i] is a vertex index that should be treated equal to vertex i . If this vector is empty, then vertices have not been unified.

The documentation for this class was generated from the following files:

- ColTopology.h
- ColTopology.cpp

8.33 VisDebug Class Reference

Functions for "visual debugging".

8.33.1 Detailed Description

Functions for "visual debugging".

You can create visual debugging objects, like lines, arrows, and planes. Objects are identified by name. If an object with that name has already been created, that object's vertices will be modified; oterwise, it will be created. Objects will be added automatically to the visdebug root, which has to be specified when creating the visdebug instance.

See also:

interactive.cpp for an example.

Todo

All the functions from Y/visdebug.c

Implementation:

These functions are instance methods (instead of global functions or class methods), so that different modules can use different visdebug roots, and, inparticular, they are thread-safe!

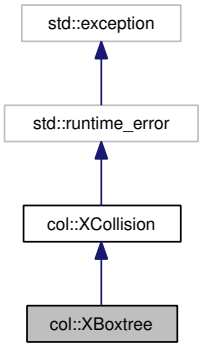
The documentation for this class was generated from the following file:

- ColVisDebug.cpp

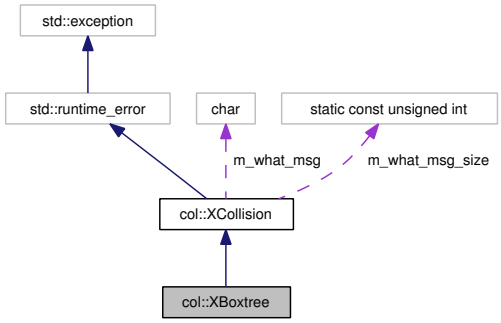
8.34 col::XBoxtree Class Reference

Will be raised by BoxTree.

Inheritance diagram for col::XBoxtree:



Collaboration diagram for col::XBoxtree:



Public Member Functions

- **XBoxtree** (const char *format,...) throw ()

8.34.1 Detailed Description

Will be raised by BoxTree.

Works exactly like XCollision.

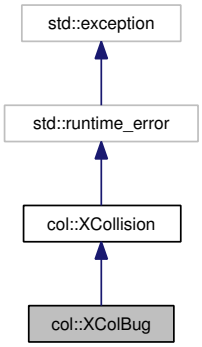
The documentation for this class was generated from the following files:

- ColExceptions.h
- ColExceptions.cpp

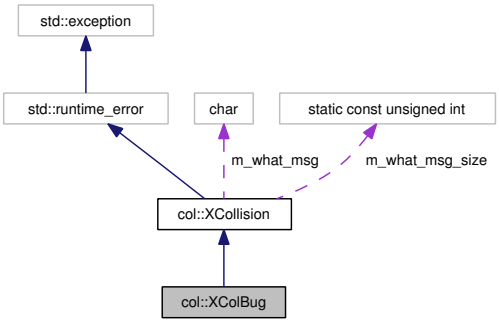
8.35 col::XColBug Class Reference

Will be raised by collision detection module, if a bug occurs somewhere in the code.

Inheritance diagram for col::XColBug:



Collaboration diagram for col::XColBug:



Public Member Functions

- **XColBug** (const char *format,...) throw ()

8.35.1 Detailed Description

Will be raised by collision detection module, if a bug occurs somewhere in the code.

Works exactly like XCollision.

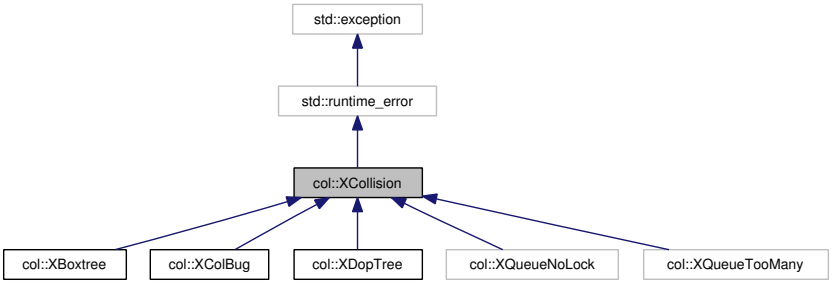
The documentation for this class was generated from the following files:

- ColExceptions.h
- ColExceptions.cpp

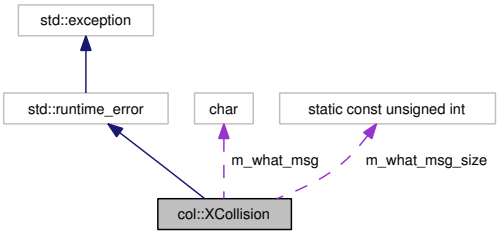
8.36 col::XCollision Class Reference

Exceptions for Collision detection module.

Inheritance diagram for col::XCollision:



Collaboration diagram for col::XCollision:



Public Member Functions

- XCollision (const char *format,...) throw ()
  Construct a collision detection exception from a format string.
- XCollision (const char *leader, const char *format,...) throw ()
  Convenience constructor for derived classes.
- void print (FILE *file=stdout) const throw ()
  Print a collision detection exception.
- void set (const char *leader, const char *format, va_list va) throw ()
  Meant for subclasses with printf-like constructors.

Protected Attributes

- char **m_what_msg** [m_what_msg_size]

Static Protected Attributes

- static const unsigned int **m_what_msg_size** = 1024

8.36.1 Detailed Description

Exceptions for Collision detection module.

Implementation:

I had to add my own message string, because std::runtime_error has only one constructor, and sometimes I can construct the message only in the body of the constructor. Does anybody know how I could've avoided that?

8.36.2 Constructor & Destructor Documentation

8.36.2.1 col::XCollision::XCollision (const char * *format*, ...) throw ()

Construct a collision detection exception from a format string.

Works like printf().

8.36.3 Member Function Documentation

8.36.3.1 void col::XCollision::set (const char * *leader*, const char * *format*, va_list *va*) throw ()

Meant for subclasses with printf-like constructors.

`va_start()` must have been done by subclass ctor! We will do `va_end` here.

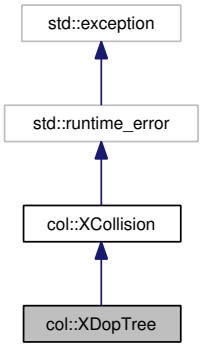
The documentation for this class was generated from the following files:

- ColExceptions.h
- ColExceptions.cpp

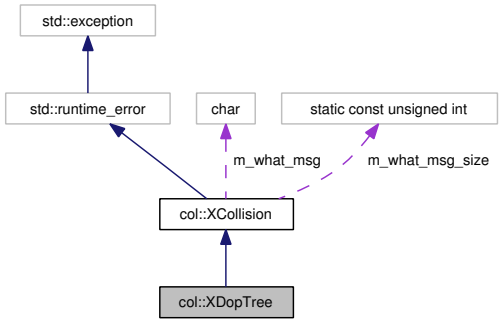
8.37 col::XDopTree Class Reference

Will be raised by DopTree.

Inheritance diagram for col::XDopTree:



Collaboration diagram for col::XDopTree:



Public Member Functions

- **XDopTree** (const char *format,...) throw ()

8.37.1 Detailed Description

Will be raised by DopTree.

Works exactly like XCollision.

The documentation for this class was generated from the following files:

- ColExceptions.h
- ColExceptions.cpp

8.38 XQueue Class Reference

Exceptions for Queue.

8.38.1 Detailed Description

Exceptions for Queue.

Works exactly like XCollision.

The documentation for this class was generated from the following file:

- ColExceptions.cpp

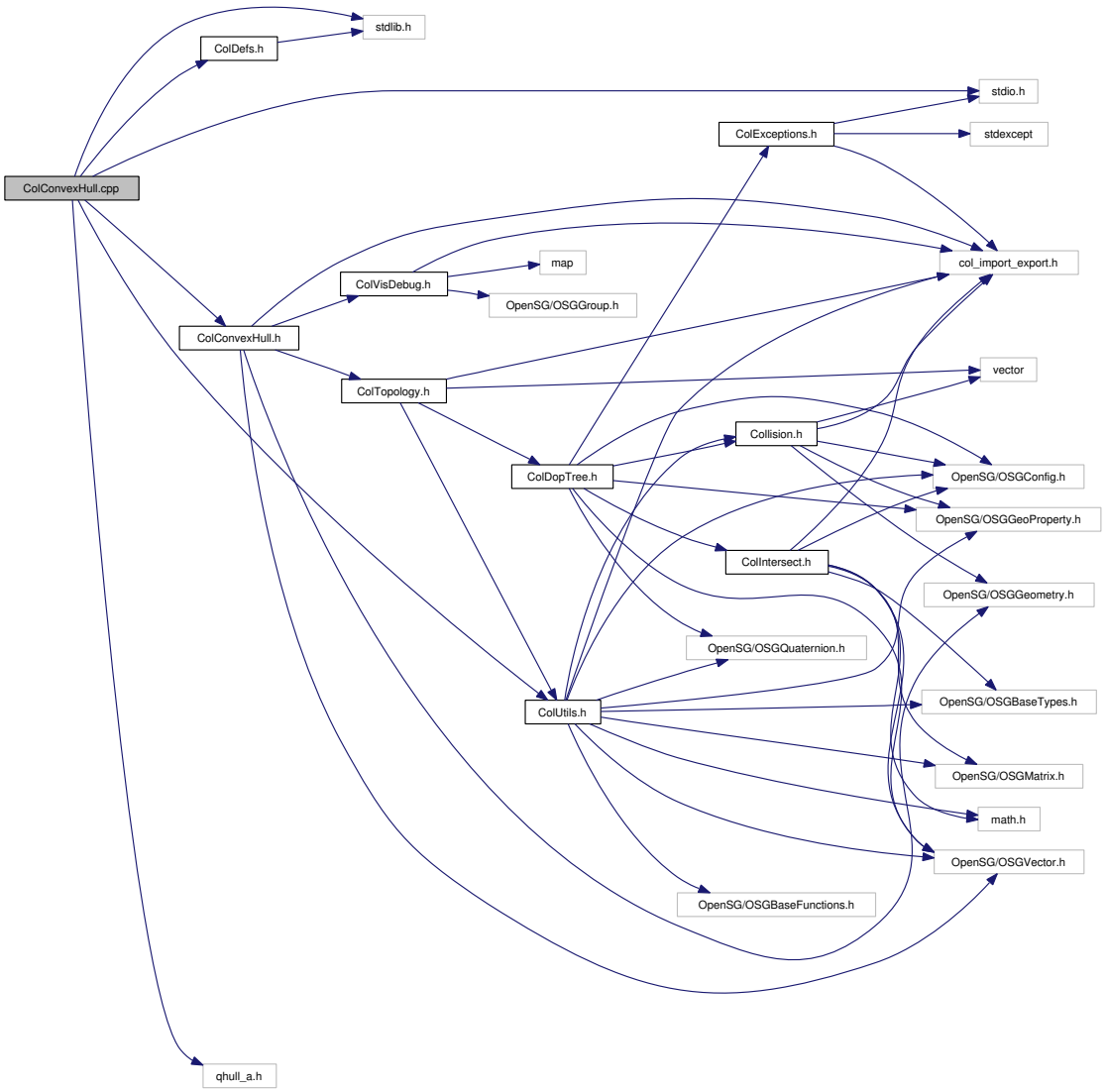
Chapter 9

CollDet File Documentation

9.1 ColConvexHull.cpp File Reference

Convex hull wrapper for qhull and collision detection of convex hulls.

Include dependency graph for ColConvexHull.cpp:



Namespaces

- namespace col

9.1.1 Detailed Description

Convex hull wrapper for qhull and collision detection of convex hulls.

Author:

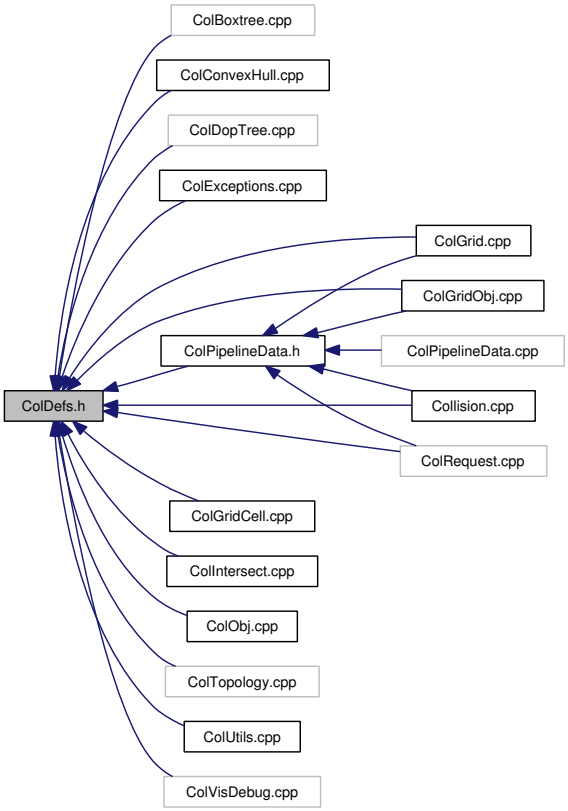
Gabriel Zachmann, Jochen Ehnes

9.2 ColDefs.h File Reference

Definitions, macros, includes, etc., needed for multi-platform compilation.

Include dependency graph for ColDefs.h:

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace col

9.2.1 Detailed Description

Definitions, macros, includes, etc., needed for multi-platform compilation.

This file should not be included in header files, in particular, it should *not* be included (directly or indirectly) in Collision.h.

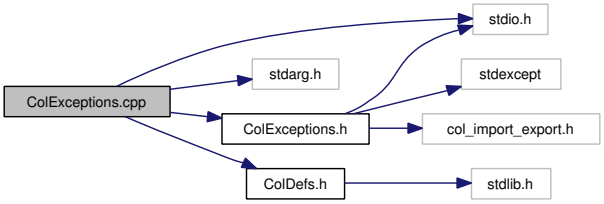
Author:

Gabriel Zachmann

9.3 ColExceptions.cpp File Reference

Exceptions which the collision detection module might throw.

Include dependency graph for ColExceptions.cpp:



Namespaces

- namespace col

9.3.1 Detailed Description

Exceptions which the collision detection module might throw.

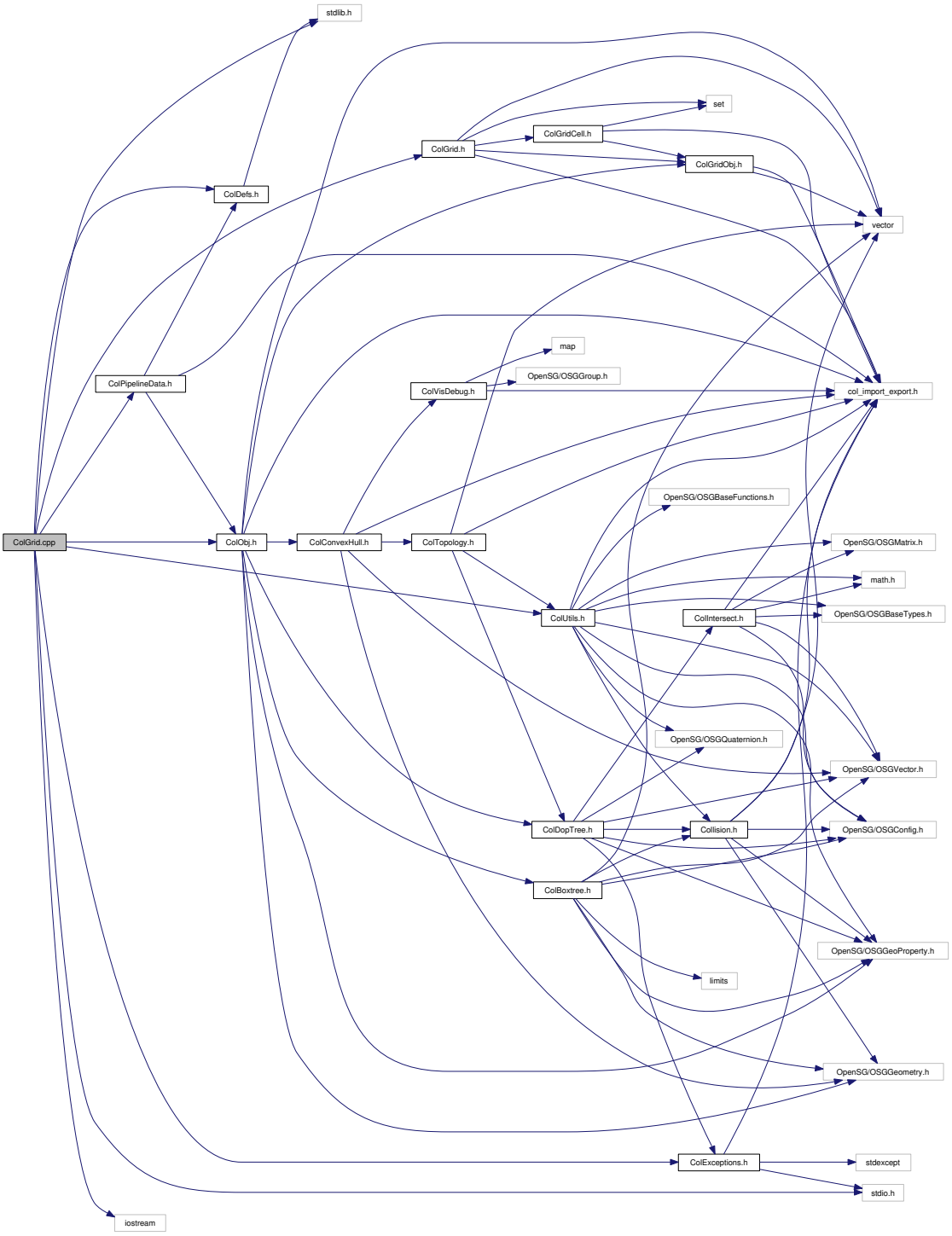
Author:

Gabriel Zachmann

9.4 ColGrid.cpp File Reference

3D grid of moving boxes

Include dependency graph for ColGrid.cpp:



Namespaces

- namespace col

9.4.1 Detailed Description

3D grid of moving boxes

Definition of the class Grid, which speeds up collision detection in conjunction with the classes GridCell and GridObj.

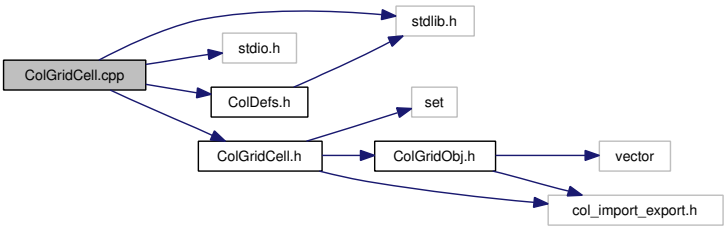
Author:

Jochen Ehnes

9.5 ColGridCell.cpp File Reference

Cells of the grid.

Include dependency graph for ColGridCell.cpp:



Namespaces

- namespace col

9.5.1 Detailed Description

Cells of the grid.

Implementation of the class GridCell

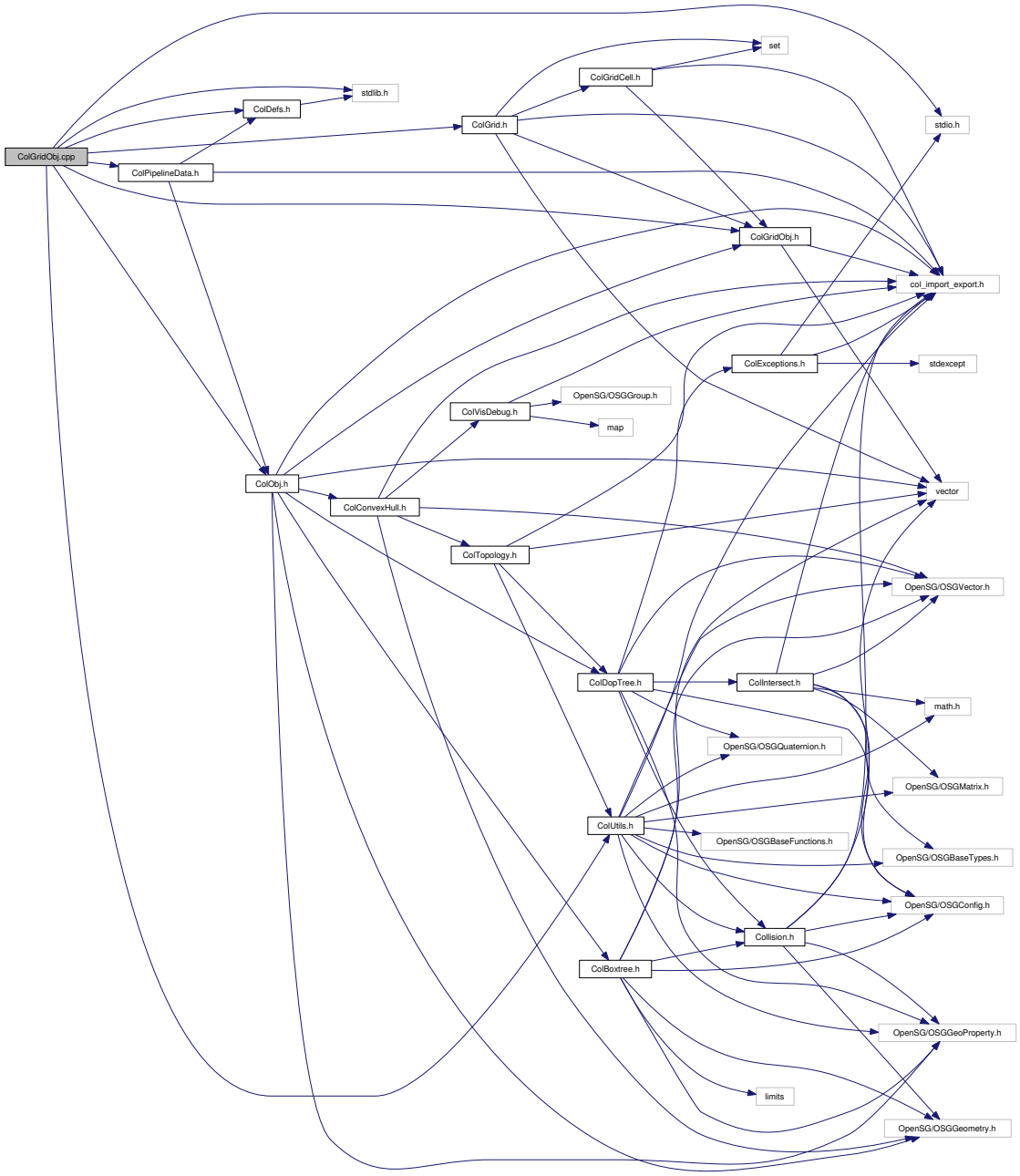
Author:

Jochen Ehnes

9.6 ColGridObj.cpp File Reference

Implementation of grid objects.

Include dependency graph for ColGridObj.cpp:



Namespaces

- namespace col

9.6.1 Detailed Description

Implementation of grid objects.

Implementation of the class GridObj. Objects of this class represent graphical objects inside a grid.

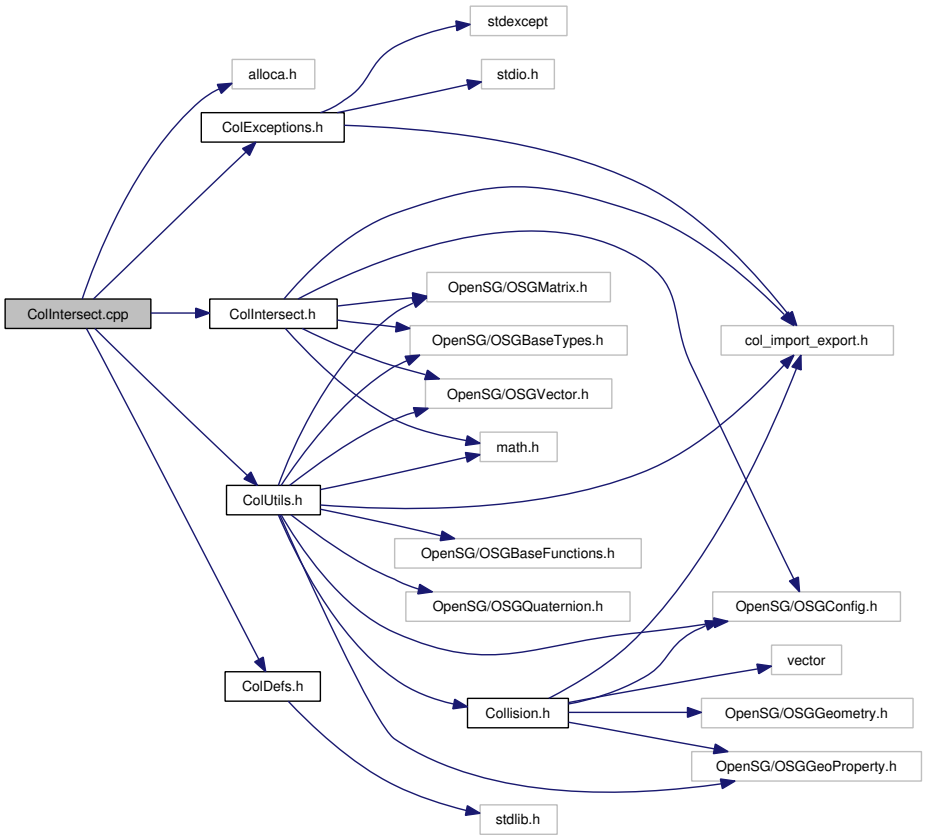
Author:

Jochen Ehnes

9.7 ColIntersect.cpp File Reference

Functions for polygon intersection testing; entry point is intersectPolygons.

Include dependency graph for ColIntersect.cpp:



Namespaces

- namespace col

Defines

- #define **COL_EXPORT**
- #define **COL_EDGE_EDGE**(__V0, __U0, __U1)
- #define **COL_EDGE_AGAINST_TRI**(_V0, _V1, _U0, _U1, _U2)

Functions

- bool **computeIntervals** (float vv0, float vv1, float vv2, float d0, float d1, float d2, float d0d1, float d0d2, float &isect0, float &isect1, float epsilon)
- bool col::intersectPolygons (const Pnt3f *poly1, int plSize1, const Pnt3f *poly2, int plSize2, const unsigned int *index1, const unsigned int *index2, const osg::Matrix *cxform)

  Check if two polygons intersect.

- bool col::intersectQuadrangles (const osg::Pnt3f &polyVv0, const osg::Pnt3f &polyVv1, const osg::Pnt3f &polyVv2, const osg::Pnt3f &polyVv3, const osg::Pnt3f &polyUv0, const osg::Pnt3f &polyUv1, const osg::Pnt3f &polyUv2, const osg::Pnt3f &polyUv3, const osg::Vec3f &normal1V, const osg::Vec3f &normal2V)

  Checks whether two quadrangles intersect.

- bool col::intersectTriangles (const Pnt3f &polyVv0, const Pnt3f &polyVv1, const Pnt3f &polyVv2, const Pnt3f &polyUv0, const Pnt3f &polyUv1, const Pnt3f &polyUv2)

  Checks if two triangles intersect.

- bool col::intersectTriangles (const Pnt3f &polyVv0, const Pnt3f &polyVv1, const Pnt3f &polyVv2, const Pnt3f &polyUv0, const Pnt3f &polyUv1, const Pnt3f &polyUv2, const Vec3f &n1V, const Vec3f &n2V)

  Checks if two triangles intersect.

- bool col::intersectCoplanarEdges (const Pnt3f &v0V, const Pnt3f &v1V, const Pnt3f &u0V, const Pnt3f &u1V, unsigned int x, unsigned int y)

  Checks if the edges intersect in 2D.

- bool col::intersectEdgePolygon (const Pnt3f &v1, const Pnt3f &v2, const Pnt3f *poly, unsigned int plSize, const Vec3f &normalV, unsigned int x, unsigned int y)

  Checks, if edge intersects polygon.

- bool col::intersectEdgePolygon (const Pnt3f &v1, const Pnt3f &v2, const Pnt3f *poly, unsigned int plSize)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- bool col::intersectArbPolygons (const Pnt3f *poly1, unsigned int plSize1, const Pnt3f *poly2, unsigned int plSize2, const Vec3f &normal1V, const Vec3f &normal2V)

  Checks if two polygons intersect.

- bool col::intersectArbPolygons (const Pnt3f *poly1, unsigned int plSize1, const Pnt3f *poly2, unsigned int plSize2)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.7.1 Detailed Description

Functions for polygon intersection testing; entry point is intersectPolygons.

Optimization:

1) For arbitrary collision testing, intersectPolygons automatically chooses the fastest algorithm, either the intersectEdgePolygon or intersectTriangles 2) intersectPolygons seems to be the fastest version for intersection tests of arbitrary polygons. If you know that you have only triangles or convex quadrangles, then it is probably faster to bypass this wrapper routine, and call intersectTriangles directly. If you know that you have only 5-gons or more vertices, then you can call intersectArbPolygons directly. 4) intersectArbPolygons: all polygons (incl. triangles) are passed directly to intersectEdgePolygon.

Todo

- Return intersection point.

Implementation:

- All "functions" beginning with "col_" are macros defined in ColUtils.h.
- Based upon routines from y/arbcoll.c and work done by Andreas Hess in 1998.

Warning:

It's assumed, that the class Pnt3f has no virtual function table! We use uninitialized memory to store an array of Pnt3f in intersectPolygons!!

Bug

Cannot handle degenerate polygons (line or point)!

Author:

Alexander Rettig (arettig@igd.fhg.de)

9.7.2 Define Documentation

9.7.2.1 #define COL_EDGE_AGAINST_TRI(_V0, _V1, _U0, _U1, _U2)

Value:

```
{                                                                   \
                                /* temporary variables, also for    */  \
                                /* interaction with COL_EDGE_EDGE   */  \
        float _Ax, _Ay, _Bx, _By, _Cx, _Cy, _EE, _DD, _FF;              \
        _Ax = _V1 [i] - _V0 [i];                                        \
        _Ay = _V1 [j] - _V0 [j];                                        \
                                /* test edge U0,U1 against V0,V1    */  \
        COL_EDGE_EDGE(_V0, _U0, _U1);                                   \
                                /* test edge U1,U2 against V0,V1    */  \
        COL_EDGE_EDGE(_V0, _U1, _U2);                                   \
                                /* test edge U2,U1 against V0,V1    */  \
        COL_EDGE_EDGE(_V0, _U2, _U0);                                   \
    }
```

9.7.2.2 #define COL_EDGE_EDGE(__V0, __U0, __U1)

Value:

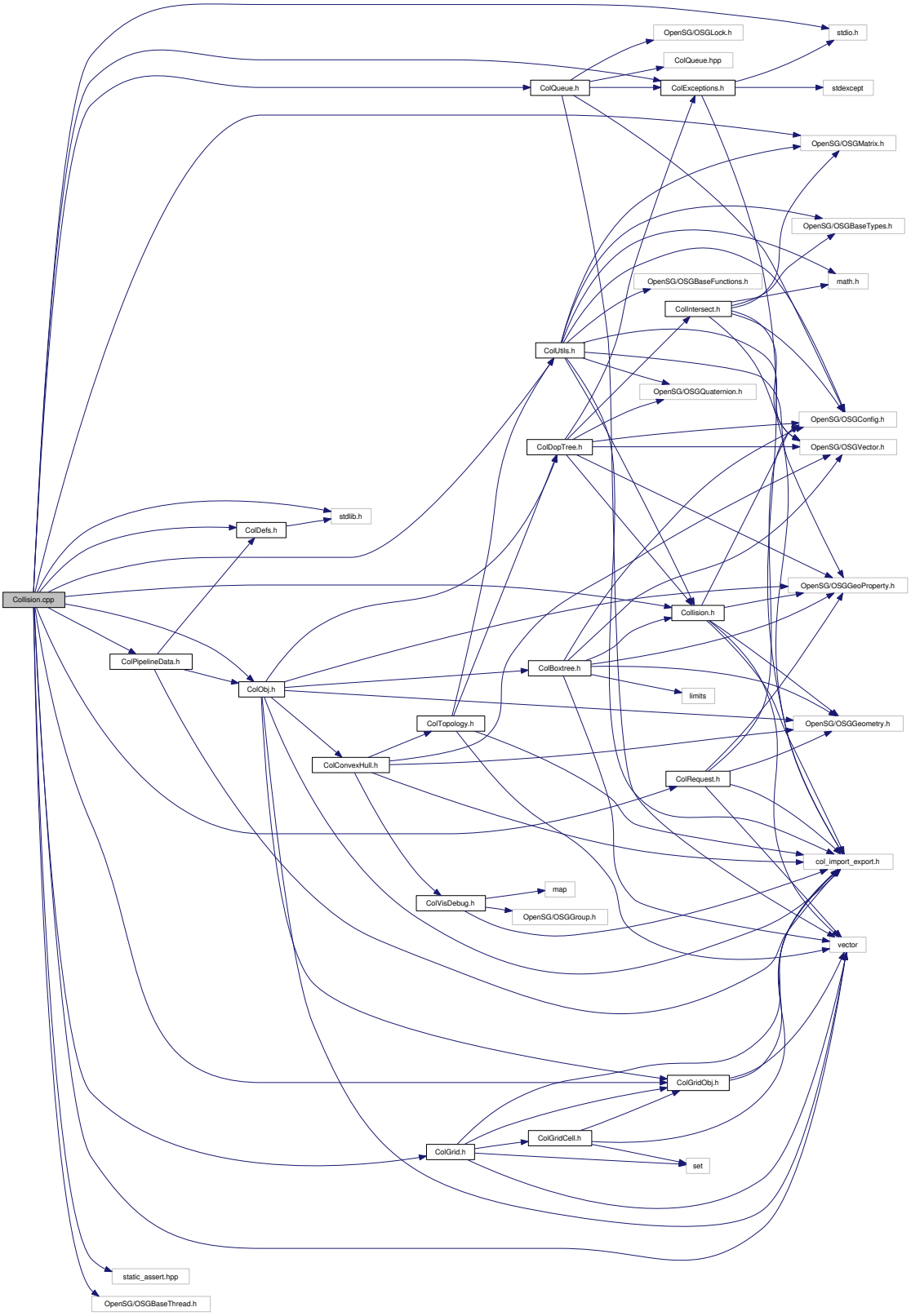
```
_Bx = __U0 [i] - __U1 [i];                                          \
        _By = __U0 [j] - __U1 [j];                                      \
        _Cx = __V0 [i] - __U0 [i];                                      \
        _Cy = __V0 [j] - __U0 [j];                                      \
        _FF = _Ay * _Bx - _Ax * _By;                                    \
        _DD = _By * _Cx - _Bx * _Cy;                                    \
        if((_FF > 0 && _DD >= 0 && _DD <= _FF) ||                       \
           (_FF < 0 && _DD <= 0 && _DD >= _FF))                         \
        {                                                               \
            _EE = _Ax * _Cy - _Ay * _Cx;                                \
            if(_FF > 0)                                                 \
```

```
        {                                                                   \
            if(_EE >= 0 && _EE <= _FF) return true;                         \
        }                                                                   \
        else                                                                \
        {                                                                   \
            if(_EE <= 0 && _EE >= _FF) return true;                         \
        }                                                                   \
    }
```

9.8 Collision.cpp File Reference

The collision detection API.

Include dependency graph for Collision.cpp:



Namespaces

- namespace col

Classes

- class **col::VtableTest_Pnt3f**
- class **col::VtableTest_Vec3f**
- class **col::VtableTest1**
- class **col::VtableTest2**

Functions

- **col::BOOST_STATIC_ASSERT** (sizeof(VtableTest1)==sizeof(VtableTest2))
- **col::BOOST_STATIC_ASSERT** (sizeof(osg::Pnt3f)!=sizeof(VtableTest_Pnt3f))
- **col::BOOST_STATIC_ASSERT** (sizeof(osg::Vec3f)!=sizeof(VtableTest_Vec3f))

9.8.1 Detailed Description

The collision detection API.

Requests to the collision detection module are made by creating a Request object containing the appropriate data, and passing that to col::request().

The C file contains the collison detection pipeline.

Precondition:

Polygons *must* be convex! (see intersectPolygons) osg::Vec3f must *not* have non-trivial constructors and desctructors, and it must *not* have a vtable, i.e., virtual functions! (see intersectPolygons)

Author:

Gabriel Zachmann

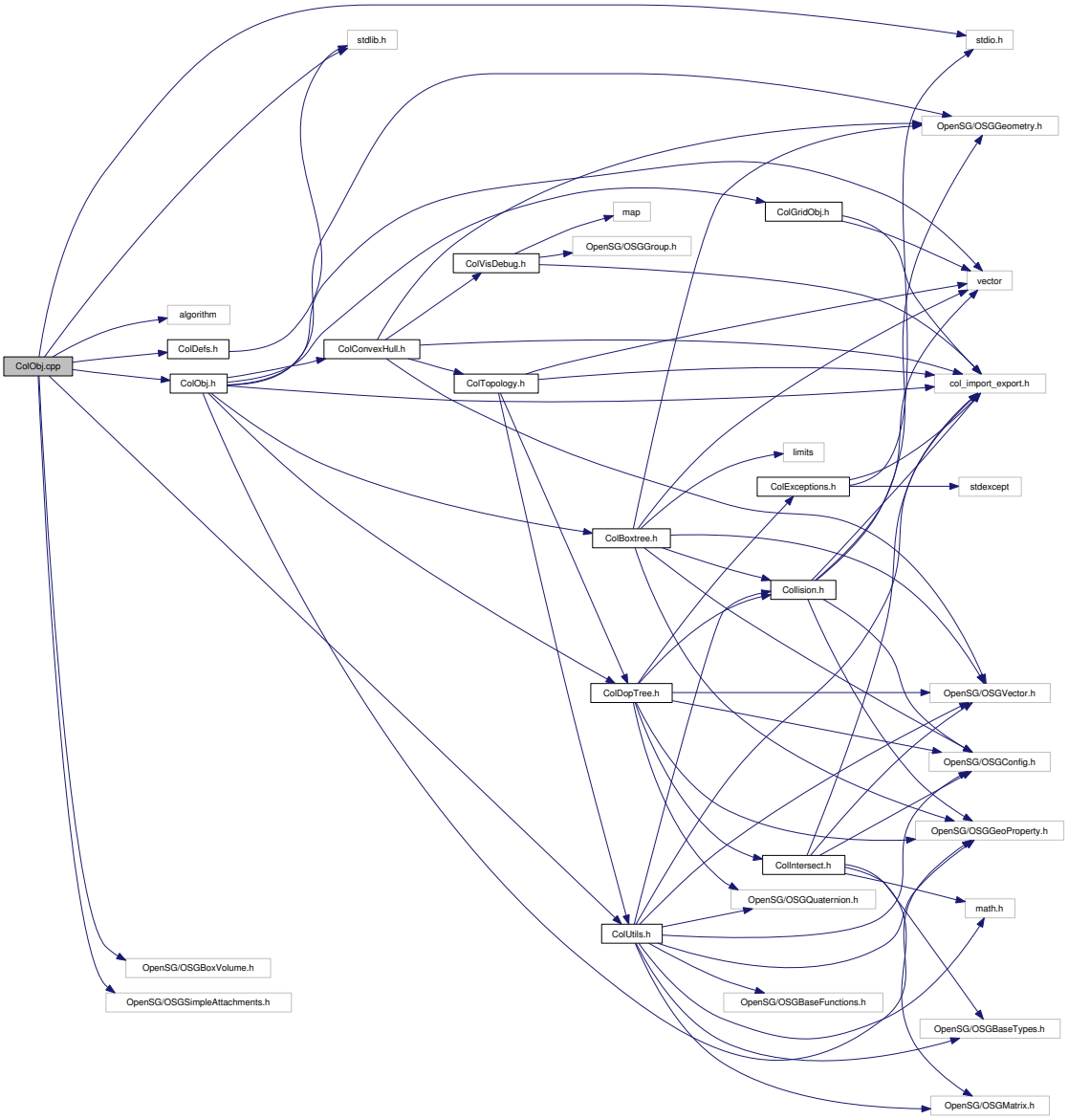
Todo

- Neg. callbacks implementieren.
- osg::Log benutzen statt stderr? (ist nicht thread-safe..)
- If there is something in the Requests queue, do a sync, *before* processing the queue.
- Instanzvariablen tatsaechlich (gemaess Guidelines) mit _ benennen.
- Threadfaehigkeit neu eingebaut, ausgiebig test (tobias)

9.9 ColObj.cpp File Reference

Infrastructure for implementing the collision detection pipeline.

Include dependency graph for ColObj.cpp:



Namespaces

- namespace col

9.9.1 Detailed Description

Infrastructure for implementing the collision detection pipeline.

Classes for storing various (possibly intermediate) information about objects and collisions.

For each object that is registered with the collision detection module a ColObj is created. This instance holds a pointer to the geometry plus various flags and auxiliary data like the convex hull, connectivity data structures, Boxtree, Doptree, etc.

Two ColObj's make a ColPair. Lists of such ColPair's are passed down the collision detection pipeline thereby filtering these lists.

For an extensive explanation of the collision detection pipeline, please see my dissertation at http://www.gabrielzachmann.org/ .

Implementation:

A word about exceptions: I have used exceptions, in particular in constructors. However, the application programmer should not need to catch exceptions, because all of them are caught by the API (at least that's the idea). One reason for this was that the app. programmer won't see any exceptions anyway, if the collision detection module runs in its own thread (I think).

Author:

Gabriel Zachmann

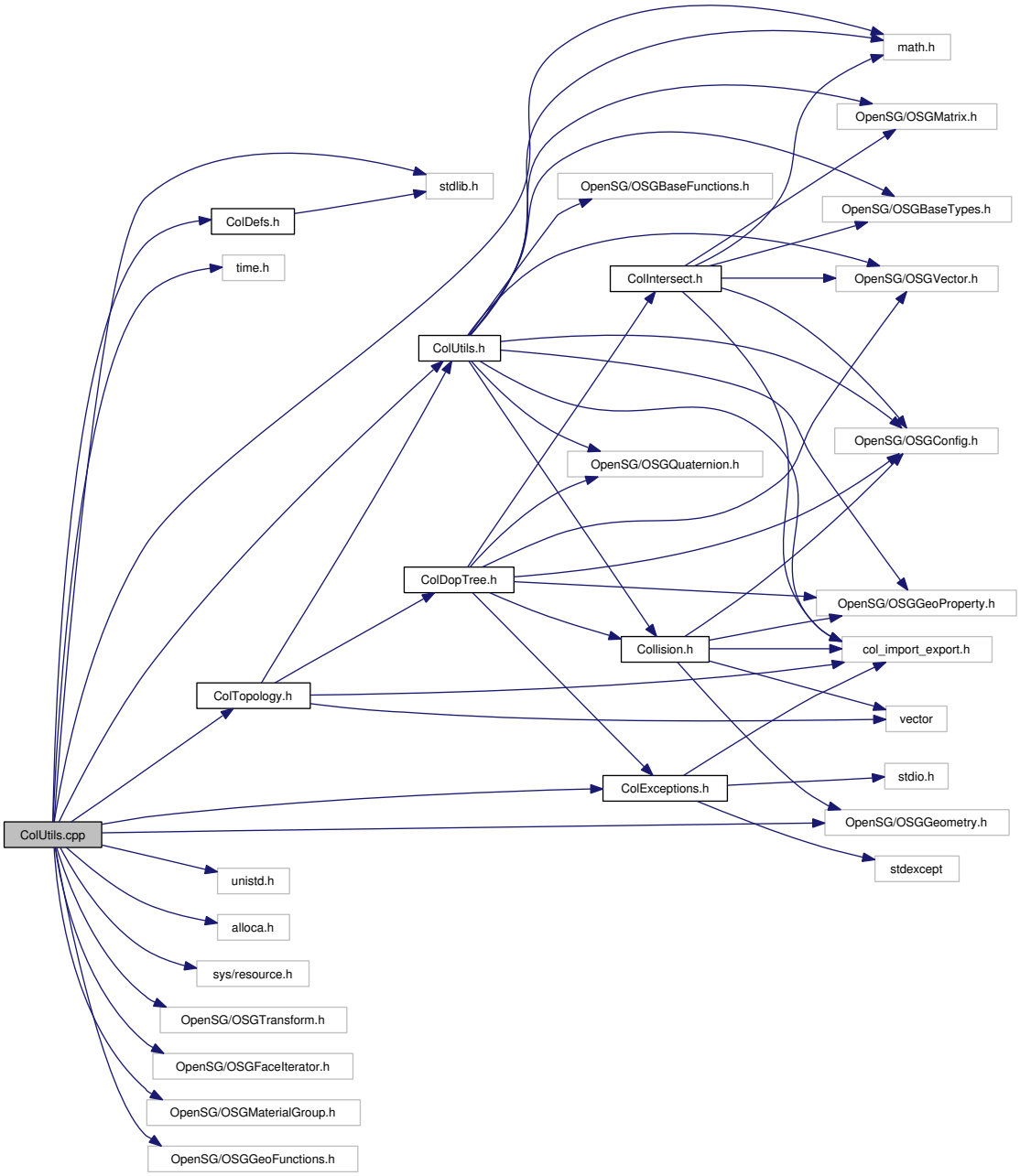
Todo

- Flags *m_stationary* and *m_flexible* (siehe ctor) auswerten.
- Die Instanzvariable *m_name* im *ColObj* zu String machen.
- Internetadresse in Kommentaren anpassen

9.10 ColUtils.cpp File Reference

Utility functions for the CollDet library. Some of them are (hopefully) temporary only, untilthey become available in OpenSG.

Include dependency graph for ColUtils.cpp:



Namespaces

- namespace col

Classes

- struct col::lessByAngle

  Compare points by angle.

- struct col::sBF

  contains some state across different invocations of addFace()

Random numbers

- #define **mod_diff**(x, y) ((x) - (y) & (M_MM-1))
- #define **is_odd**(x) ((x) & 1)
- #define **evenize**(x) ((x) & (M_MM-2))
- double col::my_drand48 (void)

  Substitute for the drand48() function under Unix (needed under Windoze).

- unsigned int col::pseudo_random (void)

  Pseudo random number generator.

- float col::pseudo_randomf (void)

  Pseudo random number generator.

Defines

- #define **COL_EXPORT**
- #define **sqr**(x) ((x)∗(x))
- #define **COL_EDGE_EDGE**(__V0, __U0, __U1)
- #define **COL_EDGE_AGAINST_TRI**(_V0, _V1, _U0, _U1, _U2)
- #define **COL_RAY_EDGE_2**(succaction)

Functions

Vector, Matrix, and Transformation Math

- float col::operator ∗ (const Vec3f &vec3, const Vec4f &vec4)

  Several 'vector ∗ vector' and 'vector ∗ point' products.

- float col::operator ∗ (const Pnt3f &pnt, const float vec[3])

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- float col::operator ∗ (const osg::Vec4f &vec4, const Pnt3f &pnt3)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- float col::operator ∗ (const Pnt3f &pnt3, const Vec3f &vec3)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- void col::operator+= (Vec4f &vec4, const Vec3f &vec3)
  Vec4f += Vec3f.

- Pnt3f col::lincomb (float c1, const Pnt3f &pnt1, float c2, const Pnt3f &pnt2)
  Affine combination of two points.

- void col::getTransfomUpto (const osg::NodePtr &cur, const osg::NodePtr &upto, osg::Matrix &result)
  Combine all transformation matrices between two nodes in the graph.

- void col::iterFaces (const osg::NodePtr &node, void(*callback)(const osg::NodePtr &, const osg::GeometryPtr &, const osg::FaceIterator &, void *), void *data)
  Calls a function for every face in the scenegraph.

- void col::countFaces (const osg::NodePtr &, const osg::GeometryPtr &, const osg::FaceIterator &, void *data)
  Count the number of faces in a scenegraph.

- float col::dist2 (const Pnt3f &pnt1, const Pnt3f &pnt2)
  Square distance between 2 points.

- float col::dist (const Pnt3f &pnt1, const Pnt3f &pnt2)
  Distance between 2 points.

- Pnt3f col::barycenter (const Pnt3f *points, const unsigned int npoints)
  Average of an array of points.

- Pnt3f col::barycenter (const vector< Pnt3f > &points)
  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- Pnt3f col::barycenter (const Pnt3f *points, const unsigned int index[ ], const unsigned int nindices)
  Average of an array of indexed points.

- Pnt3f col::barycenter (const osg::MFPnt3f *points, const unsigned int index[ ], const unsigned int nindices)
  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- Pnt3f col::barycenter (const vector< Pnt3f > &points, const TopoFace &face)
  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- bool col::collinear (const Vec3f &a, const Vec3f &b)
  Test if two vectors are collinear.

- bool col::coplanar (const Pnt3f &p0, const Pnt3f &p1, const Pnt3f &p2, const Pnt3f &q0, const Pnt3f &q1, const Pnt3f &q2)
  Test if two triangles (planes / polygons) are coplanar.

- Vec3f col::operator * (const osg::Matrix &m, const Vec3f &v)
  *Matrix * Vec3f.*

- Pnt3f col::mulM3Pnt (const osg::Matrix &m, const Pnt3f &p)

*Matrix * Pnt3f.*

- Pnt3f col::operator * (const osg::Matrix &m, const Pnt3f &p)
  *Matrix * vector.*
- osg::Matrix col::operator * (const osg::Matrix &m1, const osg::Matrix &m2)
  *Matrix * matrix.*
- Vec3f col::mulMTVec (const osg::Matrix &m, const Vec3f &v)
  *Transposed matrix * Vec3f.*
- void col::printMat (const osg::Matrix &m, FILE *file)
  Print a matrix.
- void col::printPnt (const osg::Pnt3f &p, FILE *file)
  Print a point.
- void col::dominantIndices (const Vec3f &v, unsigned int *x, unsigned int *y)
  Dominant coord plane which v is "most orthogonal" to.
- void col::dominantIndices (const Vec3f &v, unsigned int *x, unsigned int *y, unsigned int *z)
  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- unsigned int col::dominantIndex (const Vec3f &v)
  Dominant coord axis which v is "most parallel" to.
- Vec3f col::triangleNormal (const Pnt3f &p0, const Pnt3f &p1, const Pnt3f &p2)
  Normal of a triangle defined by 3 points.
- osg::Matrix col::axisToMat (const Vec3f &a, float d)
  Convert a rotation given by axis & angle to a matrix.
- unsigned int col::discretizeOri (osg::Quaternion q, unsigned int r)
  Convert an orientation (quarternion) into an integer (e.g., index).
- void col::mlerp (OSG::Matrix *intermat, const OSG::Matrix &m1, const OSG::Matrix &m2, float t)
  Matrix linear interpolation.

Geometry

- void col::sortVerticesCounterClockwise (const vector< Pnt3f > &vertex, const Vec3f &normal, TopoFace &face)
  Sort vertices of a face such that they occur counter clockwise.
- osg::NodePtr col::geomFromPoints (const vector< Pnt3f > &vertex, vector< TopoFace > &face, int gl_type, bool skip_redundant, const Vec3f normals[ ])
  Create a polyhedron from simple vertex and face arrays.
- osg::NodePtr col::geomFromPoints (const Pnt3f vertex[ ], unsigned int nvertices, unsigned int face[ ], const unsigned int face_nv[ ], unsigned int nfaces, int gl_type, bool skip_redundant, const Vec3f normals[ ])

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- osg::NodePtr col::makeCube (float radius, int gl_type)
  Create a cube as OpenSG object.
- void col::getNodeBBox (osg::NodePtr node, float min[3], float max[3])
  Get BoundingBox of an osg-node.
- osg::GeometryPtr col::getGeom (const osg::NodePtr node)
  Return the pointer to the geometry core of the node.
- osg::MFPnt3f * col::getPoints (const osg::NodePtr node)
  Return the pointer to the multi-field of the points.
- osg::MFPnt3f * col::getPoints (const osg::GeometryPtr geo)
  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- osg::GeoPositions3fPtr col::getPositions (const osg::NodePtr node)
  Return the GeoPositionsPtr of a node.
- void col::calcVertexNormals (const osg::NodePtr node, const float creaseAngle)
  Calculate vertex normals for all geometries in a subtree.
- osg::NodePtr col::findGeomNode (const osg::NodePtr node)
  Find the first node that has a geometry.
- osg::MaterialPtr col::findMaterial (const osg::NodePtr node)
  Return the material a geometry node is being drawn with.
- void col::addFace (const osg::NodePtr &node, const osg::GeometryPtr &geo, const osg::FaceIterator &face, sBF *bf)
  Add one face to a geometry/node; used by addAllFaces.
- void col::addAllFaces (const osg::NodePtr &root, sBF *bf)
  Copy all faces in the subtree into one geometry; used by mergeGeom().
- void col::mergeGeom (const osg::NodePtr &subtree, osg::NodePtr *geonode)
  Merge all geometries in a subtree into a node.

Timers, timing, sleeping, etc.

- void col::sleep (unsigned int microseconds)
  Sleep n microseconds.
- float col::time (void)
  Get the user time in milliseconds.

Floating-Point Tricks

- unsigned int col::sign (float &x)
  Returns 0 if $x < 0$, 0x80000000 otherwise.

Misc

- bool col::lockToProcessor (unsigned int processor)
  Lock the calling process to a certain processor.

Intersection Tests

- bool col::isectCoplanarTriangles (const Vec3f &normalV, const Pnt3f &polyVv0, const Pnt3f &polyVv1, const Pnt3f &polyVv2, const Pnt3f &polyUv0, const Pnt3f &polyUv1, const Pnt3f &polyUv2)
  Checks whether two coplanar triangles intersect.
- bool col::isectCoplanarEdges (const Pnt3f &v0V, const Pnt3f &v1V, const Pnt3f &u0V, const Pnt3f &u1V, unsigned int x, unsigned int y)
  Checks if the edges intersect in 2D.
- void col::isectEdgePolygon (const Pnt3f &v1, const Pnt3f &v2, const Pnt3f *poly, unsigned int plSize, const Vec3f &normalV, unsigned int x, unsigned int y, bool *isect, bool *oneside)
  Checks, if edge intersects polygon in 2D.
- void **col::isectEdgeTriangle** (const Pnt3f &v1, const Pnt3f &v2, const Pnt3f *poly, const Vec3f &normalV, unsigned int x, unsigned int y, bool *isect, bool *oneside)
- bool col::pointInPolygon (const Pnt3f &pt, const Pnt3f *poly, unsigned int plSize, unsigned int x, unsigned int y)
  Check if point is inside polygon.
- bool col::pointInTriangle (const Pnt3f &pt, const Pnt3f &v0, const Pnt3f &v1, const Pnt3f &v2, unsigned int x, unsigned int y)
  Check whether point is inside triangle.

9.10.1 Detailed Description

Utility functions for the CollDet library. Some of them are (hopefully) temporary only, untilthey become available in OpenSG.

Author:

Gabriel Zachmann

9.10.2 Define Documentation

9.10.2.1 #define COL_EDGE_AGAINST_TRI(_V0, _V1, _U0, _U1, _U2)

Value:

```
{                                                       \
                                /* temporary variables, also for    */  \
                                /* interaction with COL_EDGE_EDGE   */  \
          float _Ax, _Ay, _Bx, _By, _Cx, _Cy, _EE, _DD, _FF;            \
```

```
            _Ax = _V1 [i] - _V0 [i];                                        \
            _Ay = _V1 [j] - _V0 [j];                                        \
                                    /* test edge U0,U1 against V0,V1    */  \
            COL_EDGE_EDGE(_V0, _U0, _U1);                                   \
                                    /* test edge U1,U2 against V0,V1    */  \
            COL_EDGE_EDGE(_V0, _U1, _U2);                                   \
                                    /* test edge U2,U1 against V0,V1    */  \
            COL_EDGE_EDGE(_V0, _U2, _U0);                                   \
        }
```

9.10.2.2 #define COL_EDGE_EDGE(__V0, __U0, __U1)

Value:

```
_Bx = __U0 [i] - __U1 [i];                                              \
        _By = __U0 [j] - __U1 [j];                                      \
        _Cx = __V0 [i] - __U0 [i];                                      \
        _Cy = __V0 [j] - __U0 [j];                                      \
        _FF = _Ay * _Bx - _Ax * _By;                                    \
        _DD = _By * _Cx - _Bx * _Cy;                                    \
        if((_FF > 0 && _DD >= 0 && _DD <= _FF) ||                       \
           (_FF < 0 && _DD <= 0 && _DD >= _FF))                         \
        {                                                               \
            _EE = _Ax * _Cy - _Ay * _Cx;                                \
            if(_FF > 0)                                                 \
            {                                                           \
                if(_EE >= 0 && _EE <= _FF) return true;                 \
            }                                                           \
            else                                                        \
            {                                                           \
                if(_EE <= 0 && _EE >= _FF) return true;                 \
            }                                                           \
        }
```

9.10.2.3 #define COL_RAY_EDGE_2(succaction)

Value:

```
{                                                                       \
        if ( v1x < px && v2x < px )                                     \
            succaction;                                                 \
        v1y = v1[y];                                                    \
        v2y = v2[y];                                                    \
        if ( v1y > py && v2y > py )                                     \
            succaction;                                                 \
        if ( v1y <= py && v2y <= py )                                   \
            succaction;                                                 \
        if ( v1x >= px && v2x >= px )                                   \
        {                                                               \
            in ++ ;                                                     \
            succaction;                                                 \
        }                                                               \
        y1py = v1y - py;                                                \
        x1px = v1x - px;                                                \
        y2py = v2y - py;                                                \
        x2px = v2x - px;                                                \
        if ( y1py >= x1px && y2py >= x2px )                             \
            succaction;                                                 \
        if ( (-y1py) >= x1px && (-y2py) >= x2px )                       \
            succaction;                                                 \
                                                                        \
```

```
        y1y2 = v1y - v2y;                                               \
        t = y1py*(v2x - v1x) + x1px*y1y2;                               \
        if ( (t>0.0f && y1y2 > 0.0f) || (t<0.0f && y1y2 < 0.0f) )       \
            in ++ ;                                                     \
    }
```

Chapter 10

CollDet Page Documentation

10.1 Todo List

Class Boxtree • Die verschiedenen *MaxNVertices* konsolidieren.

Class col::BoxtreePrecomp • Es ist *nicht* egal, in welches Koord.system transformiert wird! Man sollte das abhaengig von der Anzahl der Polygone machen.

• Evtl. kann man *m_b* auch einsparen.

Class col::Callback • Option vorsehen, so dass callbacks auch aufgerufen werden, wenn keines der beiden Objekte sich bewegt hat.

• Maybe we need an additional class of Callbacks, which can be re-used for several object pairs; this would just mean, that obj1/obj2 would be overwritten by the coll.det. module for every callback actually performed.

Member col::Callback::Callback(osg::NodePtr obj1, osg::NodePtr obj2, bool always=false, bool all_polygons_in=false, *always* flag verarbeiten.

Class ColConvexHull • QHull durch CGAL ersetzen

• Den `osg::GeometryPtr` durch einen osg::GeometryConstPtr ersetzen, wenn OSG das anbietet.

• in qhull den longjmp in qh_errexit (oder so aehnlich) durch exceptions ersetzen.

• Den qhull code in einen eigenen Namespace.

Member col::CollisionPipeline::CollisionPipeline(const osg::Char8 *thread_name=NULL, unsigned int thread_id=0) Das `useHulls` Feature vereinfachen; es gibt zu viele Stellen, wo man dieses beeinflussen kann.

• Auch das *verbose* Zeugs sollte man vielleicht aufraeumen. Vielleicht einfach durch separate Fkten machen.

Member col::CollisionPipeline::check(unsigned int *num_moved=NULL) Static Variablen als Instanzvariablen der ColPipeline machen, wenn diese Funktionen hier in die Klasse ColPipeline gewandert sind. (S. Kommentar ganz oben.)

Member col::CollisionPipeline::runConcurrently(char *thread_name=NULL) Eigener Aspect. Sync mit anderen Threads. Gesynct werden muss eigtl. nur, wenn sich etwas an den Punkten oder Polygonen geaendert hat. Was ist, wenn Objekte geloescht wurden?

Class col::ColObj Was man noch tun muss ..

Member col::ColObj::ColObj(osg::GeometryPtr &geom, osg::NodePtr &node, bool flexible, bool stationary, bool comp
Parameter *m_geom* ist ueberfluessig.

Class col::ColPair Was man noch tun muss ..

Class col::Data
- Aus intersect_fun einen Funktor machen.
- Interne Daten vielleicht in eine Unterklasse ziehen.

Member col::Data::Data(const osg::NodePtr &node1, const osg::NodePtr &node2)
- Ist es ok, die Fkt getPoints() zu verwenden, wenn sich die Punkte waehrend der Koll.erkennung veraendern (durch OSG)?

Class col::Dop

Member col::Dop::Dop(const Pnt3f &pnt) Use OSG's dotprod(vec,pnt) when available.

Class col::DopNode Kann nur Dreiecke handeln! (siehe *nvertices* im header file!)

Member col::DopNode::check_down(const DopNodeList &other, Data *data, const DopTransform &dt) const
- const machen
- Check, ob pairwise processing hier in SW etwas bringt
- DopNode sollte 2 Dop's enthalten (wie der Algo es eben braucht)
- mit Intel compiler nochmal checken, ob transform2 nicht doch etwas bringt. Dito mit overlap2.
- Verbesserungen aus VRST-Paper einbauen
- In den arrays *child_other* koennte man den 2ten Zeiger einsparen, da immer other[i] und other[i]+1 betrachtet werden muessen. Dann waeren die Listen nur halb so gross.

Member col::DopNode::check_stay(const DopNodeList &other, Data *data, const Dop &e, const DopTransform &dt) c
: Per overload deklarieren.

Class DopTree
- Creation of DOP trees is *very* slow!
- Resolve all the TODO's in the code.
- Check if SECOND_ITERATION_FOR_DIAMETER really helps.
- Keine virtual methods (dtor)! (wg. Speicherplatz fuer vtable)
- DOP tree in 1 grosses Array speichern!
- Does performance increase, if all local variables are doubles?
- Zwei verschiedene DopNode's? eine Klasse fuer innere Knoten, eine fuer Blaetter; wg. Speicher fuer pgon, der in inneren Knoten nicht gebraucht wird! Evtl. kann man das auch durch union { d; pgon } machen.
- Instanzvariable *index* braucht man nicht, da das in *child*[1] gespeichert werden kann; Instanzvariable *nvertices* braucht man nicht, wenn man festlegt, dass sowieso nur 3- oder 4-ecke gespeichert werden koennen, und man pgon[3] fuer 3-ecke einfach auf *MAX_UINT* setzt.
- Klasse DopTree in Klasse DopNode mergen (oder umgekehrt).
- Verschiedene Farben bei DOP-Tree-Visualisierung.
- Code den Naming-Konventionen anpassen! (besonders m_var und M_Var)

Member col::ElemBox::set(const osg::FaceIterator &fi, const osg::MFPnt3f *points) Warum werden bei GL_QUAD_STRIP die letzten beiden Vertices *immer* geswappt??

Class col::ElemDop

Member col::ElemDop::sortindex

Member col::ElemDop::operator<(const ElemDop &other) const Sort mit ordentlichem BinaryPredicate machen! (dann ist das auch thread-safe)

Member col::Matrix::addObj(ColObj *obj) Was noch getan werden muss

Member col::Matrix::addCallback(Callback *callback, vector< ColObj > *colobjs)

Member col::Matrix::callCallbacks(const ColPair &pair) const Was noch getan werden muss

Class col::MatrixCell Flag *all_poygons* auswerten.

Member col::MatrixCell::check(bool use_hulls)
- Check whether or not only a bbox check is wanted. This would be a flag stored with each Callback, and a counter stored with the MatrixCell.
- Matrix-Inversion in ColObj::hasMoved() machen.
- Nochmal ueberpruefen, warum die berechnete Matrix m12 so stimmt; eigtl. haette ich jetzt doch eine umgekehrte Multiplikation erwartet.

- *use_hulls* in jeder MatrixCell speichern. Dann braucht man nicht das Flag global fuer alle MatrixCell's in Collision.cpp sich zu merken.

Class col::NanoTimer
- Try to estimate the overhead of a function call to start() or elapsed().
- Use PAPI (`http://icl.cs.utk.edu/projects/papi/`) or the "high resolution timers project" (`http://high-res-timers.sourceforge.net/`) when they become widely available (without kernel patches).

Class Request

Member col::Request::process(bool show_hulls, AlgoE algo, Matrix *colmatrix, std::vector< ColObj > *colobjs, std::v
Some types not yet implementated.
- process()const machen, wenn OSG erlaubt
- `show_hulls` anders (z.B. als define) implementieren

Class col::Topology
- Mehr Iterierungsfunktionen.

Member col::Topology::createRelations(void) Calc sizes of vectors first, so that we can do a resize() for each vector before filling it, in order to reduce memory fragmentation.

Class VisDebug All the functions from Y/visdebug.c

File ColIntersect.cpp
- Return intersection point.

IntersectPolygons: additional parameter for poly2's normal if available.

- Make comments consistent.
- Check comments.

Member col::intersectCoplanarEdges(const Pnt3f &v0V, const Pnt3f &v1V, const Pnt3f &u0V, const Pnt3f &u1V, unsig
Optimierung: Faktorisieren, um erste zwei Berechungen nicht mehrfach mit gleichen Parametern durchzufuehren!!!

Member col::intersectEdgePolygon(const Pnt3f &v1, const Pnt3f &v2, const Pnt3f *poly, unsigned int plSize, const Vec
Schleife ueber intersectCoplanarEdges kann optimiert werden!

Member col::intersectPolygons(const Pnt3f *poly1, int plSize1, const Pnt3f *poly2, int plSize2, const unsigned int *inde
Da ein Viereck planar ist, braucht man eigentlich die Unterscheidung zwischen Quadrangle und Quadstrip doch nicht machen, oder?

File Collision.cpp
- Neg. callbacks implementieren.

- osg::Log benutzen statt stderr? (ist nicht thread-safe..)
- If there is something in the Requests queue, do a sync, *before* processing the queue.
- Instanzvariablen tatsaechlich (gemaess Guidelines) mit _ benennen.
- Threadfaehigkeit neu eingebaut, ausgiebig test (tobias)

Member col::PolyIntersectT(Data *data) Als Funktor machen!

File ColObj.cpp
- Flags *m_stationary* and *m_flexible* (siehe ctor) auswerten.
- Die Instanzvariable *m_name* im *ColObj* zu String machen.
- Internetadresse in Kommentaren anpassen

Member col::pseudo_random(void)
- Check that the seed is not the unique "bad" number as explained in http://home.t-online.de/home/mok-kong.shen/ .
- *a* should be a primitive root of *c* (see URL above).
- Groesseres *c* suchen.

Member col::axisToMat(const Vec3f &a, float d) d = -d; kann man wahrscheinlich wieder rauswerfen, wenn man unten die Transposition auch entfernt.

Member col::sortVerticesCounterClockwise(const vector< Pnt3f > &vertex, const Vec3f &normal, TopoFace &face) Use "Lamda Library" (Boost).

Member col::geomFromPoints(const Pnt3f vertex[], unsigned int nvertices, unsigned int face[], const unsigned int face_ immer noch wird die Variable NumOri gebraucht..

Member col::sleep(unsigned int microseconds)
- Funktion suchen, die Mikrosekunden kann.

Member col::lockToProcessor(unsigned int processor) Implement for Windows.

Member col::isectEdgePolygon(const Pnt3f &v1, const Pnt3f &v2, const Pnt3f *poly, unsigned int plSize, const Vec3f & Schleife ueber intersectCoplanarEdges koennte optimiert werden.

Member col::pointInPolygon(const Pnt3f &pt, const Pnt3f *poly, unsigned int plSize, unsigned int x, unsigned int y) Fuer Dreiecke und Vierecke optimieren!

10.2 Bug List

Member col::ColObj::updateBBox(void) Funktioniert noch nicht, da OSG einen Bug hat.

Class DopTree

Member col::ElemBox::operator==(const ElemBox &other) const Does not work if the polygons are the same but the start index is different!

Class col::FibRand The range has not really been checked/verified.

Class Grid The *Grid* class and friends are probably not multi-thread-safe, i.e., several threads asking the same grid for a list of intersecting boxes will get different (wrong) answers. (This is because of the cycle counters.)

Member col::Matrix::addObj(ColObj *obj) Bekannte Bugs dieser Funktion

Member col::Matrix::callCallbacks(const ColPair &pair) const Bekannte Bugs dieser Funktion

Member col::Topology::createFromGeom(const osg::GeometryPtr geom, bool unify=false, float tolerance=NearZero) Es kommen immer 0 faces raus! hat FaceIterator einen Bug?!

File ColIntersect.cpp Cannot handle degenerate polygons (line or point)!

Member col::pseudo_random(void) Not multithread-safe.

Member col::discretizeOri(osg::Quaternion q, unsigned int r) I think, that if two rotations yield the same index, then they represent "close" rotations - but I haven't checked yet. (Note that the reverse statement is not true.)

Member col::sleep(unsigned int microseconds) On most platforms (Windows, Linux, single-CPU SGI), this function will sleep at least 10 millliseconds! (On Linux, usleep and nanosleep don't work as advertised, as of RedHat 7.2)

Index