GHOST SDK Reference Manual

April 29, 2002

Chapter GHOST SDK Reference Manual

Chapter I - GHOST Library

Chapter 1. Core Classes

class gstNode

Summary

#include “gstNode.h”
class gstNode ;

Description

Base class for scene graph nodes.

Public constructors

virtual ~gstNode() ;
Destructor.

Public Members

virtual gstTransform* AsTransform() ;
Casting.

virtual gstNode* Clone() const = 0;
Clone.

virtual gstNode* getByName(const gstNodeName& name) ;
Returns first node in subtree with nodeName = name. Otherwise, returns NULL.

gstBoolean getInSceneGraph() const;
Returns TRUE if node is in scene graph.

virtual gstNodeName getName() const;
Returns name of node.

virtual gstType getTypeId() const;
Virtual form of getClasstypeId.

virtual gstBoolean isOfType(gstType type) const;
Static: Return TRUE if class is of the given type or is derived from that type.

virtual void putInSceneGraph() ;
For extension: Called when object is put in scene graph.

virtual void removeFromSceneGraph() ;
For extension: Called when object is removed from scene graph.

virtual void setName(const gstNodeName& name) ;
Set name of node.

static gstType getClassTypeId() ;
Static: get type id of this class.

static gstBoolean staticIsOfType(gstType type) ;
Virtual form of staticIsOfType.

Protected constructors

gstNode() ;
This class is intended as a base class only, the constructors are protected so that instances can not be created.

gstNode(const gstNode& origNode) ;

		gstNode(const gstNode* origNode) ;
Protected Data	static gstType	gstNodeClassTypeId
	gstBoolean	inSceneGraph
	gstNodeName	nodeName

class gstNodeName

Summary

#include “gstNodeName.h”
class gstNodeName ;

Description

Handles character string for node names.

Public constructors

gstNodeName() ;
Constructor.

gstNodeName(const char* newName) ;
Constructor.

gstNodeName(const gstNodeName& newName) ;
Constructor.

gstNodeName(gstNodeName* newName) ;
Constructor.

~gstNodeName() ;
Destructor.

Public Operators

gstNodeName& operator=(const char* p) ;
Assignment operator.

gstNodeName& operator=(const gstNodeName& p) ;
Assignment operator.

gstNodeName& operator=(const gstNodeName* p) ;
Assignment operator.

operator char*() ;
Cast to string operator.

gstBoolean operator!=(const gstNodeName& p) const;
Inequality operator.

gstBoolean operator<(const gstNodeName& p) const;
Less than operator.

gstBoolean operator<=(const gstNodeName& p) const;
Less than or equal to operator.

gstBoolean operator==(const gstNodeName& p) const;
Equality operator.

gstBoolean operator>(const gstNodeName& p) const;
Greater than operator.

gstBoolean operator>=(const gstNodeName& p) const;
Greater than or equal to operator.

Protected Data	char*	name

class gstPHANToM

Summary

#include “gstPHANToM.h”
class gstPHANToM : public gstDynamic;

Description

PHANToM haptic interface class. This class is used to set and access all the basic state of the PHANToM haptic interface. State information includes: 1) whether or not forces are enabled; 2) the position of the PHANToM; 3) attached boundary classes; 4) PHANToM/object collision information; and 4) whether or not the PHANToM is in the haptic scene. Two types of PHANToM position, as well as two reference frames for these positions, are available. There is a local reference frame (the default) and a world coordinate reference frame (used by any methods appended with "WC"). Information on both the "real" position of the PHANToM as well as information on a Surface Contact Point (SCP) projection are available. NOTE: The PHANToM encoders are reset by default upon creation of a gstPHANToM object. For more information on reseting the PHANToM, refer to the section of the GHOST Programming Guide pertaining to the gstPHANToM class.

Public constructors

gstPHANToM(char* configFile ,int resetEncoders = TRUE) ;
Constructor. Requires character string indicating name of the PHANToM initialization file. ResetEncoders specifies if PHANToM encoders are to be reset to zero when creating instance. Encoders will be reset if TRUE, otherwise encoders are not reset. Default value is TRUE.

virtual ~gstPHANToM() ;
Destructor.

Public Members

gstBoolean addedCollision() ;
For internal use.

gstBoundaryCube* attachCubeBoundary() ;
Attaches a cube boundary object to the PHANToM using the information retrieved from gstPHANToMInfoStruct Note: Make sure the PHANToM has already been added to the scene graph and has a parent node, or else this routine will not succeed.

gstBoundaryCube* attachMaximalBoundary() ;
Attaches a maximally sized boundary object to the PHANToM using the information retrieved from gstPHANToMInfoStruct.

gstBoundary* detachBoundary() ;
Detaches the current boundary from the PHANToM and removes it from the scene.

virtual int forcesOff() ;
For internal use.

virtual int forcesOn() ;
For internal use.

double getAverageUpdateRate() ;
Get average servo-loop (PHANToM update) rate [Hz].

gstBoundary* getBoundaryObj() ;
Get currently attached boundary object.

int getCalibrationStatus() ;

For internal use.

gstCollisionInfoStruct* getCollisionInfo() ;
For internal use.

gstCollisionInfoStruct* getCurrentCollisionInfo() ;
For internal use.

double getDeltaT() ;
Get the time increment between the last two PHANToM updates [seconds].

gstTransform* getDynamicCollisionObj() ;
For internal use.

gstBoolean getDynamicFriction() const;
Used internally to keep track of friction state.

gstEffect* getEffect() const;
Return current effect currently associated with the PHANToM, or NULL if none exists.

gstBoolean getForceOutput() const;
Returns TRUE if forces are to be used during simulation. Default is TRUE.

gstVector getGimbalAngles() ;
Returns rotation angles of encoder gimbal in radians. Angles are relative to PHANToM tip not to base ref frame.

int getHardwareRevision() ;
Get internal hardware rev number of phantom. Only supported for desktop phantom.

const gstPHANToMInfoStruct* getInfo() ;
Provides PHANToM setup info. Contains information about the PHANToM configuration, like: is6DOF, isDesktop, and PHANToM specific workspace dimensions.

gstCollisionInfoStruct* getLastCollisionInfo() ;
For internal use.

gstPoint getLastFrictionSCP() const;
Used internally to track SCP under various conditions.

double getLastKavg() ;

int getLastNumCollisionObjs() ;

void getLastPosition_WC(gstPoint& pt) ;
Get previous position of PHANToM in world coordinates (i.e., position before previous call to gstPHANToM::update()).

void getLastSCP_WC(gstPoint& _lastSCP_WC) ;
For extension: Get previous position of PHANToM SCP in world coordinates (i.e. Position before previous call to gstPHANToM::update().

void getLastSCP_WC(gstPoint& _lastSCP_WC ,gstPoint& _lastSCPBeforeDynamicMove_WC) ;
For internal use.

gstManipulator* getManipulator() const;
Returns current manipulator currently associated with the PHANToM.

double getMaxGain() const;
Different PHANTOM models can simulate different material stiffnesses without causing buzzing or other artifacts. The gain here is such that when multiplied by a normalized stiffness of 1.0 it represent that max material spring constant the PHANTOM can simulate.

void getMotorOverheatState(float motorTemps [ ]) ;
Gets estimated motor temperatures where 0 is room temperature and 1.0 is max temperature before overheat. Takes array of 6 floats to get temperature for each motor.

gstPHANToM* getNextPHANToM() const;
Gets next PHANToM in the scene graph.

int getNumCollisionObjs() ;

virtual gstPoint getPosition_WC() ;
Get position of PHANToM in world coordinates. The origin of this object's local reference frame is actually the PHANToM position.

virtual void getPosition_WC(gstPoint& pt) ;
Get position of PHANToM in world coordinates. The origin of this object's local reference frame is actually the PHANToM position.

virtual gstVector getReactionForce_WC() ;
Get reaction force in world reference frame from PHANToM [Newtons].

virtual gstVector getReactionTorque_WC() ;
Get reaction torque in world reference frame from PHANToM [Newton*millimeters].

gstPHANToM_SCP* getSCPNode() ;
Get pointer of SCP node.

void getSCP_P(gstPoint& pt) ;
Get current Surface Contact Point (SCP) in parent reference frame.

void getSCP_WC(gstPoint& pt) ;
Get current Surface Contact Point (SCP) in world coordinates.

gstBoolean getStatus() ;
Returns whether there is a dev fault.

gstBoolean getStylusPresence() const;
Returns TRUE if stylus presence sensor is on, otherwise FALSE. If no stylus is attached, output is undefined.

gstBoolean getStylusSwitch() const;
Returns TRUE if stylus switch is depressed, otherwise FALSE. If no stylus is attached, output is undefined.

virtual gstType getTypeId() const;
Virtual form of getClassTypeId.

gstBoolean getValidConstruction() const;

Returns TRUE is instance had no errors during construction.

virtual void invalidateCumTransf() ;
For extension: Used by system or for creating sub-classes only. When this object or any object above it in the scene changes its local transformMatrix, then this node's cumulative transform matrix is not valid until it is recomputed based on the new data. This function invalidates the cumulative transform matrix for this node and its inverse.

virtual gstBoolean isOfType(gstType type) const;
Virtual form of staticIsOfType.

virtual void prepareToUpdateGraphics() ;
For extension: Used by system or for creating sub-classes only. This function prepares data to be sent to a graphics callback. When gstScene::updateGraphics() is called by the application, gstScene stalls the application process and--in the haptic process--calls this method for each node in the scene that has had graphics information changes since the last call to gstScene::updateGraphics(). When finished, the application process continues by calling updateGraphics for all the same nodes. UpdateGraphics() actually calls the user graphics callback with the current graphics information that was copied over in the calls to this method (prepareToUpdateGraphics()). The haptics process, therefore, is ONLY used to copy the current graphics information and the application process calls the callback functions.
Each subclass of gstShape that passes additional data to this graphics callback must redefine this method and call <PARENTCLASS>::prepareToUpdateGraphics() before exiting. In order to pass additional data to graphics callback, cbData must point to the new datatype that adds any additional fields. These fields are then filled in by prepareToUpdateGraphics().

virtual void putInSceneGraph() ;
For extension: Call when object is added to scene graph.

virtual void removeFromSceneGraph() ;
For extension: Called when object is removed from scene graph.

void resetEncoders() ;
Resets the encoders on the PHANToM device.

void setBoundaryObj(gstBoundary* newBoundaryObj) ;
Attach a boundary object (e.g. An object of gstBoundary type such as gstBoundaryCube) to the PHANToM. The gstBoundary node should be IN the scene graph, but only this instance of gstPHANToM will be able to feel the boundary object.

void setDynamicFriction(gstBoolean dynFriction) ;

void setEffect(gstEffect* newEffect) ;
Associate an special effect (i.e. From gstEffect class and sub-classes) with the PHANToM. If a previous effect was in place, it is stopped before the new effect is added.

virtual int setForce(const gstVector& force ,const gstVector& torque) ;
For internal use.

void setForceOutput(gstBoolean flag) ;
If flag is TRUE then forces will be turned on and sent to PHANToM when the simulation is active (i.e. The servo loop is running). Otherwise, forces will not be turned on nor used at any time.

virtual int setForce_WC(const gstVector& forceArg_WC ,const gstVector& torqueArg_WC) ;

For internal use.

void setLastFrictionSCP(const gstPoint& lastFrictionSCP) ;

void setLastKavg(double kavg) ;

void setLastSCP_WC(gstPoint& newLastSCP_WC) ;
For internal use.

void setManipulator(gstManipulator* newManipulator) ;
Associate a manipulator (i.e. From gstManipulator class and sub-classes) with the PHANToM. If a previous manipulator exists, it is stopped before the new manipulator is added.

void setMaxGain(double newMaxGain) ;
See above for info about max gain. The set function should normally not be used since each PHANTOM model has a correct max gain value as part of its configuration.

void setSCPNode(gstPHANToM_SCP* newSCPNode) ;
Set pointer to an SCP node. The position of newSCPNode will be set to the current SCP location every call to gstPHANToM::update().

void setSCP_WC(const gstPoint& newSCP_WC) ;
For internal use.

gstBoolean startEffect() ;
Start the effect (if any) associated with the PHANToM.

gstBoolean startManipulator() ;
Start the manipulator (if any) associated with the PHANToM.

gstBoolean stopEffect() ;
Stop the effect (if any) associated with the PHANToM.

gstBoolean stopManipulator() ;
Stop the manipulator (if any) associated with the PHANToM.

int update() ;
For internal use.

void updateCalibration() ;
For internal use.

static int IsResetNeeded(const char** configFiles ,int num) ;
For internal use.

static int disableAllForces() ;
For internal use.

static int enableAllForces() ;
For internal use.

static gstType getClassTypeId() ;
Static: get type id of this class.

static gstPHANToM* getPHANToMsInScene() ;
Returns the first PHANToM in the scene graph. Use getNextPHANToM to retrieve next PHANToM in the scene graph.

static gstBoolean staticIsOfType(gstType type) ;
Return TRUE if class is of the given type or is derived from that type.

static gstBoolean statusOfAll() ;
For internal use.

static int updateAll() ;
For internal use.

Public Data

static int maxCollisions - For internal use.
gstPoint scene_SCP - For internal use.

Protected members

void prepareForNextLoop() ;
For internal use.

void resetAutoCalibration() ;
For internal use.

void resetError() ;
For internal use.

Protected Data

gstVector PHANToMForce
gstVector PHANToMTorque
gstPHANToM_SCP* SCPNode
gstPoint SCP_DCOC
gstPoint SCP, SCP_WC
gstBoolean _useForces
gstBoundary* boundaryObj
gstCollisionInfoStruct collisionInfo [MAX_COLLISIONS]
gstBoolean d_dynamicFriction
gstPoint d_lastFrictionSCP
gstDynamic* dynamicCollisionObj
gstEffect* effect
gstBoolean forcesAreOn
gstVector gimbalAngles
static gstPHANToM* gstPHANToMHead
gstPHANToMInfoStruct info
gstCollisionInfoStruct lastCollisionInfo [MAX_COLLISIONS]
double lastKavg
int lastNumCollisionObjs
gstPoint lastPos
gstPoint lastPosition_WC
gstPoint lastSCPBeforeDynamicMove_WC
gstPoint lastSCP_WC
gstManipulator* manipulator
double maxGain
gstPHANToM* nextPHANToM
int numCollisionObjs

int	phantomId
gstBoolean	resetSceneSCP
gstBoolean	stylusPresence
gstBoolean	stylusSwitch
gstBoolean	validConstruction

class gstPHANToM_SCP

Summary

#include “gstPHANToM_SCP.h”
class gstPHANToM_SCP : public gstShape;

Description

Represents the PHANToM's SCP in the scene graph, usually not needed. GstPHANToM node's Surface Contact Point (SCP): If the PHANToM is in contact with a surface, the SCP is the calculated point of contact on the contacted surface. Otherwise, the SCP coincides with the position of the PHANToM. Note: This class is only mean't to be used as a convenience node. This allows a separate node for the SCP and another for the gstPHANToM position. You query the same state from just the gstPHANToM node.

Public constructors

gstPHANToM_SCP() ;
Constructor.

gstPHANToM_SCP(const gstPHANToM_SCP& origPHANToM_SCPNode) ;
Constructor.

gstPHANToM_SCP(const gstPHANToM_SCP* origPHANToM_SCPNode) ;
Constructor.

virtual ~gstPHANToM_SCP() ;
Destructor.

Public Members

virtual gstNode* Clone() const;
Clone.

gstPHANToM_SCP* ClonePHANToM_SCP() const;

virtual gstType getTypeId() const;
Virtual form of getClassTypeId.

virtual gstBoolean isOfType(gstType type) const;
Virtual form of staticIsOfType.

virtual void putInSceneGraph() ;
For extension: Used by system or for creating sub-classes only. Called when object is added to scene graph.

virtual void removeFromSceneGraph() ;
For extension: Used by system or for creating sub-classes only. Called when object is removed from scene graph.

static gstType getClassTypeId() ;
Static: get type id of this class.

static gstBoolean staticIsOfType(gstType type) ;
Return TRUE if class is of the given type or is derived from that type.

class gstScene

Summary

#include “gstScene.h”
class gstScene ;

Description

Holds haptic scene and mediates interaction with the PHANToM and servo loop.

Public Constants

SMOOTHING_OFF
SMOOTHING_ON

Public constructors

gstScene() ;
Constructor.

~gstScene() ;
Performs a recursive delete on all nodes under the root.

Public Members

void cleanupServoLoop() ;
Clean up the haptic simulation after an internal error.

int getDoneOneLoop() const;
For internal use.

int getDoneServoLoop() const;
Returns TRUE if haptics process has finished (i.e. If the servo loop is not running).

gstTransform* getRoot() ;

const gstTransform* getRoot() const;
Get root node of haptic scene graph.

gstBoolean getSafety() const;
Returns TRUE if safety limits are on.

double getSafetyLimit() const;
Returns the currently set duty cycle limit (sec).

int getSmoothing() const;
Get smoothing level of the scene.

gstBoolean lock() ;
Lock() a gstScene before making changes to it if the servo loop is running. NOTE: Changes to the phantom or its ancestor nodes are not supported when the servo loop is running, even if the scene is locked. Also, effects and manipulators will still be run even if the scene is locked, so if this is incompatible with the changes being made they should be stopped before making changes. Finally, force fields are not enabled when the scene is locked. Returns TRUE if lock acquired, FALSE otherwise (if servo loop stopped).

int postServoLoop() ;

int preServoLoop() ;

int servoLoop() ;

For internal use.

void setDoneOneLoop(gstBoolean newVal) ;
For internal use.

void setPostServoCallback(gstServoCallback* pCallback ,void* pUserData) ;
Provide a callback to be called in the servoloop thread following the last servoloop tick.

void setPreServoCallback(gstServoCallback* pCallback ,void* pUserData) ;
Provide a callback to be called in the servoloop thread before the first servoloop tick.

void setRoot(gstTransform* newRoot) ;
Set root node of haptic scene graph.

void setSafety(gstBoolean _s) ;
Turns safety on and off. If safety is off, the servo loop is allowed to take as much CPU time as necessary to finish. This may result in instability or crashes. The default is TRUE (on).

void setSafetyLimit(double limit) ;
Sets the duty cycle limit in seconds that gets checked by the safety.

void setServoCallback(gstServoCallback* pCallback ,void* pUserData) ;
In addition to the standard Ghost servoloop activity, you can hook in your own callback function that will get called at servo rate.

void setSmoothing(int newLevel) ;
Set smoothing level for scene. Smoothing attempts to remove high-frequency variations in gstPolyMesh geometries. Currently only 0 (OFF) and 1 (ON) are supported.

int startServoLoop() ;
Start the haptic simulation as separate process. Control is returned immediately.

void stopServoLoop() ;
Stop the haptic simulation.

void turnForcesOff() ;
For internal use.

void turnForcesOn() ;
For internal use.

void unlock() ;
Call unlock() after a lock() to resume normal servo loop behavior. NOTE: the scene should remain locked for only a few servo loop iterations (each iteration is 1ms) to maintain an accurate haptic simulation.

void updateEvents() ;
Calls the event callback for all nodes in the scene which have new events since the last call to updateEvents(). Each callback is called once for each new event.

void updateGraphics() ;
Calls the graphics callbacks for all nodes in the scene which have changed since the last call to updateGraphics(). This means nodes which have not changed will not have their graphics callback called.

Protected Data

double Kfriction
gstBoolean checkStatus
int doneServoLoop, doneOneLoop
double dutyCycleLimit
gstBoolean dutyCycleSafety
int dynamicF
int firstLoop, forcesOn
gstBoolean needLock, locked
void* m_pPostServoCBUserData
gstServoCallback* m_pPostServoCallback
void* m_pPreServoCBUserData
gstServoCallback* m_pPreServoCallback
void* m_pServoCBUserData
gstServoCallback* m_pServoCallback - These callback function pointers can be set to extend the standard hooks into the servoloop thread.
int preparingGraphics, preparingEvents
gstTransform* rootNode
int smoothingLevel
double dynamicFrictionAvg, staticFrictionAvg, surfaceDampingAvg

class gstSeparator

Summary

#include “gstSeparator.h”
class gstSeparator : public gstTransform;

Description

Node class that allows grouping of nodes under it into a sub-tree.

Public constructors

gstSeparator() ;
Constructor.

gstSeparator(const gstSeparator& origSeparatorNode) ;
Constructor.

gstSeparator(const gstSeparator* origSeparatorNode) ;
Constructor.

virtual ~gstSeparator() ;
Destructor.

Public Members

virtual gstSeparator* AsSeparator() ;
Casting.

virtual gstNode* Clone() const;
Clone.

gstSeparator* CloneSeparator() const;

virtual void addChild(gstTransform* newChild) ;
Add child to separator.

virtual double getBoundingRadiusOfChildren() ;

virtual gstNode* getByName(const gstNodeName& name) ;
Returns first node in subtree with nodeName = name, otherwise returns NULL.

gstTransform* getChild(int childIndex) ;
Returns a pointer to the child indicated by childIndex. If childIndex is invalid, an error is sent to gstErrorHandler and NULL is returned. Note that childIndex is valid from 0 to (number_of_childern - 1).

int getNumChildren() const;
Returns the number of children under the separator.

int getNumChildrenDEBUG() ;
For internal use.

virtual gstPoint getScaleFactor() const;
Get scale factors along scale orientation axis. If the matrix has not been set explicity, then the scale orientation axis' coincide with the local reference frame axis'.

virtual void getScaleFactor(gstPoint& newScale) const;
Get scale factors along scale orientation axis. If the matrix has not been set explicity, then the scale orientation

axis' coincide with the local reference frame axis'.

virtual gstType getTypeId() const;
Virtual form of getClassTypeId.

virtual void invalidateCumTouchability() ;
Mark the cumulative touchability invalid for this node and all children beneath this separator.

virtual void invalidateCumTransf() ;
For extension: Used by system or for creating sub-classes only. When this object or any object above it in the scene changes its local transformMatrix, then this node's cumulative transform matrix is not valid until it is recomputed based on the new data. This function invalidates the cumulative transform matrix for this node and its inverse.

virtual void invalidateCumTransfAndMakeUntouched() ;
For internal use.

virtual gstBoolean isOfType(gstType type) const;
Virtual form of staticIsOfType.

virtual void prepareToUpdateGraphics() ;
For extension: Used by system or for creating sub-classes only. This function prepares data to be sent to a graphics callback. When gstScene::updateGraphics() is called by the application, gstScene stalls the application process and--in the haptic process--calls this method for each node in the scene that has had graphics information changes since the last call to gstScene::updateGraphics(). When finished, the application process continues by calling updateGraphics for all the same nodes. UpdateGraphics() actually calls the user graphics callback with the current graphics information that was copied over in the calls to this method (prepareToUpdateGraphics()). The haptics process, therefore, is ONLY used to copy the current graphics information and the application process calls the callback functions.
Each subclass of gstShape that passes additional data to this graphics callback must redefine this method and call <PARENTCLASS>::prepareToUpdateGraphics() before exiting. In order to pass additional data to graphics callback, cbData must point to the new datatype that adds any additional fields. These fields are then filled in by prepareToUpdateGraphics().

virtual void putInSceneGraph() ;
For extension: Used by system or for creating sub-classes only. Called when object is added to scene graph.

virtual void removeChild(gstTransform* childToRemove) ;
If "childToRemove" exists, it is removed from the separator. Otherwise, if it is invalid, an error is sent to gstErrorHandler.

virtual gstTransform* removeChild(int childIndex) ;
If "childIndex" is valid, the appropriate child is removed and a pointer to the child is returned. If "childIndex" is invalid, an error is sent to gstErrorHandler and NULL is returned. Note that "childIndex" is valid from 0 to (number_of_children - 1).

virtual void removeFromSceneGraph() ;
For extension: Used by system or for creating sub-classes only. Called when object is removed from scene graph.

virtual void rotate(const gstVector& axis ,double rad) ;
Accumulate rotation with previous rotation for the separator.

virtual void scale(double scale) ;

Accumulate scale with previous scale for the separator.

virtual void setCenter(const gstPoint& newCenter) ;
Set center for the separator.

virtual void setDynamicDependent(gstTransform* newDynamicDep) ;
For internal use.

virtual void setPosition(const gstPoint& newPos) ;
Overwrite previous position of node with new position.

virtual void setPosition(double x ,
double y ,
double z) ;
Overwrite previous position of node with new position.

virtual void setRotate(const gstVector& axis ,double rad) ;
DEPRECATED: Name changed for consistency.

virtual void setRotation(const gstVector& axis ,double rad) ;
Overwrite previous rotation with new rotation.

virtual void setScale(double newScale) ;
Overwrite previous scale with new scale.

virtual void setTouchableByPHANToM(const gstBoolean bTouchable) ;
If flag is FALSE, then all gstShape descendents of this node will become untouchable by virtue of parent/child cumulative touchability. If the flag is TRUE, then it is up to the child nodes to determine their touchability.

virtual void setTransformMatrix(const gstTransformMatrix& matrix) ;
Set homogenous transformation matrix for the separator.

virtual void setTranslate(const gstPoint& translation) ;
DEPRECATED: Name changed for consistency.

virtual void setTranslate(double x ,
double y ,
double z) ;
DEPRECATED: Name changed for consistency.

virtual void setTranslation(const gstPoint& translation) ;
Overwrite previous translation with new translation.

virtual void setTranslation(double x ,
double y ,
double z) ;
Overwrite previous translation with new translation.

virtual void translate(const gstPoint& translation) ;
Accumulate translation with previous translation for the separator.

virtual void translate(double x ,
double y ,
double z) ;

Accumulate translation with previous translation for the separator.

static gstType getClassTypeId() ;
Static: get type id of this class.

static gstBoolean staticIsOfType(gstType type) ;
Return TRUE if class is of the given type or is derived from that type.

Protected Data

gstNodeList*	children
int	numChildren

class gstTransform

Summary

#include “gstTransform.h”
class gstTransform : public gstNode;

Description

Node class that adds 3D transformations and callbacks to nodes.

Public constructors

virtual ~gstTransform() ;
Destructor.

Public Members

virtual gstSeparator* AsSeparator() ;

virtual gstTransform* AsTransform() ;
Casting.

virtual gstNode* Clone() const;
Clone.

gstTransform* CloneTransform() const;

virtual gstPoint fromParent(const gstPoint& p) ;
Transform point "p", which is in the parent coordinate reference frame, to the point in the local coordinate reference frame.

virtual gstVector fromParent(const gstVector& v) ;
Transform vector "v", which is in the parent coordinate reference frame, to the vector in the local coordinate reference frame.

virtual gstPoint fromWorld(const gstPoint& p) ;
Transform point "p", which is in the world coordinate reference frame, to the point in the local coordinate reference frame.

virtual gstVector fromWorld(const gstVector& v) ;
Transform vector "v", which is in the world coordinate reference frame, to the vector in the local coordinate reference frame.

virtual gstPoint getCenter() ;
Get x,y,z coordinates of center. Not supported for gstShape classes.

virtual void getCenter(gstPoint& centerArg) const;
Get x,y,z coordinates of center. Not supported for gstShape classes.

gstTransformMatrix getCumTransformMatrixDEBUG() ;
For internal use.

virtual gstTransformMatrix& getCumulativeTransform() ;

```
For extension:
 Get cumulative transformation matrix.
 NOTE: result is a non constant reference.
```

gstTransformMatrix getCumulativeTransformMatrix() ;
Get cumulative transformation matrix.

void getCumulativeTransformMatrix(gstTransformMatrix& matrixArg) ;
Get cumulative transformation matrix.

gstTransform* getDynamicDependent() const;
Get dynamic dependent. A node should only have one ancestor of type gstDynamic, this method returns that node.

void* getGraphicsCBUserData() const;
Get graphics callback user data.

gstBoolean getInGraphicsQueue() ;
For internal use.

gstTransformMatrix getObjectTransformMatrixDEBUG() ;
For internal use.

gstTransform* getParent() const;
Returns parent of node in graph. Returns NULL if the node has no parent or is the root of the scene graph.

virtual gstPoint getPosition() ;
Get position in local coordinate reference frame.

virtual void getPosition(gstPoint& pos) ;
Get position in local coordinate reference frame.

virtual gstPoint getPosition_WC() ;
Get x,y,z translation in world coordinates.

virtual void getPosition_WC(gstPoint& pos) ;
Get x,y,z translation in world coordinates.

virtual void getRotation(gstVector& axisArg ,double* radArg) const;
For internal use.

gstPoint getRotationAngles() const;
Get equivilant rotation of current rotation matrix (orientation) based on successive rotations around x,y,z axes. Angles are in radians and use right hand rule.

void getRotationAngles(gstPoint& axes) const;
Get equivilant rotation of current rotation matrix (orientation) based on successive rotations around x,y,z axes. Angles are in radians and use right hand rule.

gstTransformMatrix getRotationMatrix() ;
Get rotation matrix.

void getRotationMatrix(gstTransformMatrix& matrixArg) ;
Get rotation matrix.

virtual gstPoint getScaleFactor() const;
Get x,y,z scale factors along scale orientation axis.

virtual void getScaleFactor(gstPoint& scaleFactorArg) const;
Get x,y,z scale factors along scale orientation axis.

gstTransformMatrix getScaleOrientationMatrix() ;
Get scale orientation matrix.

void getScaleOrientationMatrix(gstTransformMatrix& matrixArg) ;
Get scale orientation matrix.

gstBoolean getTouchableByPHANToM() const;
Returns the locally stored touchability state. This is not the same as the cumulative isTouchableByPHANToM().

gstTransformMatrix getTransformMatrix() ;
Get homogenous transformation matrix.

void getTransformMatrix(gstTransformMatrix& matrixArg) ;
Get homogenous transformation matrix.

virtual void getTranslation(gstPoint& translationValue) const;
Get translation in local coordinate reference frame.

virtual void getTranslation_WC(gstPoint& translationValue) const;
Get translation in world coordinate reference frame.

virtual gstType getTypeId() const;
Virtual form of getClassTypeId.

virtual void getWorldTranslation(gstPoint& translationValue) ;
For internal use.

virtual void invalidateCumTouchability() ;
Marks the cumulative touchability state as invalid for this node.

virtual void invalidateCumTransf() ;
For extension: Used by system or for creating sub-classes only. Invalidate cumulative transformation matrix. When this object or any object above it in the scene changes its local transformMatrix, this node's cumulative transform matrix becomes not valid until it is recomputed based on the new data. This function invalidates the cumulative transform matrix for this node and its inverse.

virtual void invalidateCumTransfAndMakeUntouched() ;
For internal use.

virtual gstBoolean isOfType(gstType type) const;
Virtual form of staticIsOfType.

gstBoolean isTouchableByPHANToM() ;
Will evaluate the cumulative touchability state of the node based on its own touchability as well as its parent nodes.

virtual void makeUntouched() ;
For internal use.

virtual void prepareToUpdateEvents() ;

For internal use.

virtual void prepareToUpdateGraphics() ;
For extension: Used by system or for creating sub-classes only. Set up data structures to update graphics.

virtual void putInSceneGraph() ;
For extension: Used by system or for creating sub-classes only. Called when object is put in scene graph.

virtual void removeFromSceneGraph() ;
For extension: Used by system or for creating sub-classes only. Called when object is removed from scene graph.

virtual void rotate(const gstVector& axis ,double rad) ;
DEPRECATED: Name changed for consistency.

void rotateLM(const gstVector& axis ,double radians) ;
Add new rotation specified by vector/axis angle approach to existing rotation. Angle is in radians. This version left multiplies: newRot = givenRot * currentRot.

void rotateRM(const gstVector& axis ,double radians) ;
Add new rotation specified by vector/axis angle approach to existing rotation. Angle is in radians. This version right multiplies: newRot = currentRot * givenRot.

virtual void scale(const gstPoint& newScale) ;
Accumulate scale with previous scale of node.

virtual void scale(double scale) ;
Accumulate uniform scale with previous scale of node.

virtual void scale(double x ,
double y ,
double z) ;
Accumualte scale with previous scale of node.

virtual void setCenter(const gstPoint& newCenter) ;
Overwrite previous center position of node with new center position. Not supported for gstShape classes.

virtual void setDynamicDependent(gstTransform* newDynamicDep) ;
For internal use.

void setEventCallback(gstEventCallback* callback ,void* userdata) ;
Set event callback. "callback" points to a user callback function that is called when gstScene::updateEvents is called by the application and this instance of gstTransform or its subclasses have new event information. Event information is passed as the second parameter "callbackData" and should be cast to type (gstEvent *). The fields of this structure are interpreted differently by each class and you should consult the GHOST Programming Guide for an list of nodes and their interpretation of these fields for various events.

void setGraphicsCallback(gstGraphicsCallback* callback ,void* userdata) ;
Set graphics callback. "callback" points to a user callback function that is called when gstScene::updateGraphics is called by the application and this instance of gstTransform or its subclasses have new graphics information. Graphics information is passed as the second parameter "callbackData" and should be cast to the type (CLASSNAMEgraphicsCBData *) for the correct class of this instance.

void setParent(gstTransform* newParent) ;

For internal use.

virtual void setPosition(const gstPoint& newPos) ;
Overwrite previous translation with new translation.

virtual void setPosition(double x ,
double y ,
double z) ;
Overwrite previous translation with new translation.

virtual void setPosition_WC(const gstPoint& newPos_WC) ;
Overwrite previous translation with new translation given as a position in world reference frame coordinates.

virtual void setRotate(const gstVector& axis ,double rad) ;
DEPRECATED: Name changed for consistency.

virtual void setRotation(const gstVector& axis ,double rad) ;
Overwrite previous rotation of node using vector/angle method [radians].

virtual void setScale(const gstPoint& newScale) ;
Overwrite previous scale of node with new scale.

virtual void setScale(double newScale) ;
Overwrite previous scale of node with new uniform scale.

virtual void setScale(double x ,
double y ,
double z) ;
Overwrite previous scale of node with new scale.

virtual void setTouchableByPHANToM(const gstBoolean bTouchable) ;
Modify the touchability state of this node. When a node is marked as untouchable, then its collision detection should respect that and not introduce any forces.

void setTransformMatrix(const gstTransformMatrix& matrix) ;
Sets homogenous transformation matrix. Note: Any call to setTransformMatrix resets all scale rotate, and translate values set previously. Conversly, any call to setRotation, setScale or setTranslation resets the previous call to setTransform. Setting a matrix explicitly using this method or any of the array accessors to set a specific entry of the 4x4 matrix will cause this transform matrix to become "User Defined". A user defined matrix ceases to have the CSRTC' composite form described in the GHOST Programming Guide. Instead, no composite form is assumed and some operations may run slower with the "User Defined" matrix since some optimizations are not performed.

virtual void setTranslate(const gstPoint& translation) ;
DEPRECATED: Name changed for consistency.

virtual void setTranslate(double x ,
double y ,
double z) ;
DEPRECATED: Name changed for consistency.

virtual void setTranslation(const gstPoint& translation) ;
Set translation.

virtual void setTranslation(double x ,
double y ,
double z) ;
Overwrite previous translation with new translation.

virtual gstPoint toParent(const gstPoint& p) ;
Transform point "p", which is in the local coordinate reference frame, to the point in the parent coordinate reference frame.

virtual gstVector toParent(const gstVector& v) ;
Transform vector "v", which is in the local coordinate reference frame, to the vector in the parent coordinate reference frame.

virtual gstPoint toWorld(const gstPoint& p) ;
Transform point "p", which is in the local coordinate reference frame, to the point in the world coordinate reference frame.

virtual gstVector toWorld(const gstVector& v) ;
Transform vector "v", which is in the local coordinate reference frame, to the vector in the world coordinate reference frame.

virtual void translate(const gstPoint& translation) ;
Accumulate translation with previous translation of node.

virtual void translate(double x ,
double y ,
double z) ;
Accumulate translation with previous translation of node.

void updateEvents() ;
For internal use.

void updateGraphics() ;
For internal use.

static gstType getClassTypeId() ;
Static: get type id of this class.

static gstBoolean staticIsOfType(gstType type) ;
Return TRUE if class is of the given type or is derived from that type.

static void staticPrepareToUpdateEvents() ;
For internal use.

static void staticPrepareToUpdateGraphics() ;
For internal use.

static void staticUpdateEvents() ;
For internal use.

static void staticUpdateGraphics() ;
For internal use.

Protected constructors

gstTransform() ;
This class is intended as a base class only, the constructors are protected so that instances can not be created.

gstTransform(const gstTransform& origTransfNode) ;
Constructor.

gstTransform(const gstTransform* origTransfNode) ;
Constructor.

Protected members

gstBoolean addEvent(const gstEvent& newEvent) ;
Useful methods.

void addToGraphicsQueue() ;

Protected Data

gstBoolean changedThisServoLoop
int cumTransformValid
gstTransform* dynamicDependent
void* eventCBUserData
gstEventCallback* eventCallback
static gstTransform* eventsQueueHead
static gstTransform* eventsWaitingToBeFiredHead
void* graphicsCBData - This is used to store graphics data for callback. Memory is allocated in this class's constructor.
void* graphicsCBUserData
gstGraphicsCallback* graphicsCallback
static gstTransform* graphicsQueueHead
gstBoolean inEventQueue
gstBoolean inGraphicsQueue - Event and Graphics Queues data.
gstTransformMatrix cumTransf, lastCumTransf
gstTransformMatrix objTransf, lastObjTransf - Transform matrices.
gstTransform* nextEventInQueue
gstTransform* nextEventWaitingToBeFired
gstTransform* nextNodeInGraphicsQueue
gstTransform* nextWaitingForGraphicsCallback
gstTransform* parent
gstVector scaleFactor
static gstTransform* waitingForGraphicsCallbackHead

class gstTransformMatrix

Summary

#include “gstTransformMatrix.h”
class gstTransformMatrix ;

Description

Homogeneous 3d tranformation matrix class. Order of operations is center, scale, rotation, then translation unless matrix is user defined. A matrix becomes user defined if one or more entries of the 4x4 array have been set explicitly. If a matrix is user defined then no assumptions are made about its form.

Public constructors

gstTransformMatrix() ;
Constructor.

gstTransformMatrix(const double* mat) ;
Constructor from array of 16 values stored by row.

gstTransformMatrix(const gstBasicTransformMatrix& mat) ;
Constructor from gstBasicTransformMatrix.

```
gstTransformMatrix(double a11 ,
                                                    double a12 ,
                                                    double a13 ,
                                                    double a21 ,
                                                    double a22 ,
                                                    double a23 ,
                                                    double a31 ,
                                                    double a32 ,
                                                    double a33 ) ;
```

Constructor (makes matrix userDefined).

```
gstTransformMatrix(double a11 ,
                                                    double a12 ,
                                                    double a13 ,
                                                    double a14 ,
                                                    double a21 ,
                                                    double a22 ,
                                                    double a23 ,
                                                    double a24 ,
                                                    double a31 ,
                                                    double a32 ,
                                                    double a33 ,
                                                    double a34 ,
                                                    double a41 ,
                                                    double a42 ,
                                                    double a43 ,
                                                    double a44 ) ;
```

Constructor (makes matrix userDefined).

Public Operators

bool	operator!=(const gstTransformMatrix& rhs) const;
const double&	operator()(const int i ,const int j) const;

double& operator()(const int i ,const int j) ;
Operator() get element (i,j).

bool operator==(const gstTransformMatrix& rhs) const;

__forceinline Proxy1D operator[](const int i) ;

__forceinline const Proxy1D operator[](const int i) const;

Public Members

bool compare(const gstTransformMatrix& rhs) const;
Same as operator==.

gstPoint fromLocal(const gstPoint& p) ;
Transform point from local reference frame to parent reference frame.

gstVector fromLocal(const gstVector& v) ;
Transform point from local reference frame to parent reference frame. Preserves vector length.

double get(int i ,int j) const;
Get value of row "i" and column "j" of matrix.

gstBasicTransformMatrix getBasicTransformMatrix() const;
Get the matrix as a gstBasicTransformMatrix.

gstPoint getCenter() const;
Get x,y,z coordinates of center.

void getCenter(gstPoint& centerArg) const;
Get x,y,z coordinates of center.

gstTransformMatrix getInverse(bool& success) ;

gstTransformMatrix getInverse() ;
Get inverse of matrix. 'success' is set to TRUE if the matrix was inverted (i.e. Has an inverse), and FALSE if the matrix inverse failed (e.g. The matrix was singular). The return value is indeterminate if success is FALSE. Calling getInverse without the 'success' variable will also yield indeterminate results if no inverse for the matrix exists.

const double* getOpenGLMatrix() const;
Get opengl matrix, for passing to glLoadMatrix for example.

gstPoint getRotationAngles() const;
Get equivalent rotation based on successive rotations around x,y,z axes.

void getRotationAngles(gstPoint& axes) const;
Get equivilant rotation of current rotation matrix (orientation) based on successive rotations around x,y,z axes. Angles are in radians and use right hand rule. The values returned can be positive or negative rotation angles relative to 0. If you need to compare rotation values, you should first convert them all to positive rotations. For instance:
Axes.setx(axes.x() < 0 ? 2 * M_PI + axes.x() : axes.x()); axes.sety(axes.y() < 0 ? 2 * M_PI + axes.y() : axes.y()); axes.setz(axes.z() < 0 ? 2 * M_PI + axes.z() : axes.z());

gstTransformMatrix getRotationMatrix() const;

Get rotation matrix.

void getRotationMatrix(double transfMat [3] [3]) const;
Get 3x3 rotation matrix.

void getRotationMatrix(gstTransformMatrix& transfMat) const;
Get 3x3 rotation matrix.

gstPoint getScaleFactor() const;
Get scale factors along axis' defined by scale orientation.

void getScaleFactor(gstPoint& scaleFactor) const;
Get scale factors along axis' defined by scale orientation.

void getScaleOrientationMatrix(gstTransformMatrix& transfMat) const;
Get 3x3 scale orientation matrix.

gstPoint getTranslation() const;
Get x,y,z translation.

void getTranslation(gstPoint& translationArg) const;
Get x,y,z translation.

gstTransformMatrix getTranspose() const;
Get transpose of matrix.

int getUserDefined() const;
Returns TRUE if user has explicitely set matrix (i.e. By setting elements in the matrix directly instead of using "setCenter", "setScale", "setRotation", or "setTranslation" operations). "getCenter", "getScale", "getRotation", and "getTranslation" operations will not return valid information if a matrix is user defined and will cause a GST_TRANSFORM_RESET_ERROR.

void identity() ;
Sets matrix to identity matrix.

void inverse() ;
DEPRECATED: Name changed for consistency.

bool invert() ;
Sets value of matrix to inverse. Returns TRUE if the matrix was successfully inverted, FALSE if there is no inverse to the matrix (e.g. The matrix is singular). The matrix is not modified (i.e. Is unchanged as a result of calling the function) if the function returns FALSE.

gstBoolean isIdentity() const;
Returns TRUE if the matrix is the identity matrix.

void printSelf() ;
Print values of matrix.

void resetFields() ;
For internal use.

void rotate(const gstVector& v ,double a) ;
DEPRECATED: Name change for consistency.

void rotateLM(const gstVector& axis ,double radians) ;
Add new rotation specified by vector/axis angle approach to existing rotation. Angle is in radians. This version left multiplies: newRot = givenRot * currentRot.

void rotateRM(const gstVector& axis ,double radians) ;
Add new rotation specified by vector/axis angle approach to existing rotation. Angle is in radians. This version right multiplies: newRot = currentRot * givenRot.

void scale(const gstPoint& newScale) ;
Add "newScale" to existing scale.

void scale(double argX ,
double argY ,
double argZ) ;
Add new x,y,z components to existing scale,.

void set(const double mat [4] [4]) ;
(makes matrix userDefined).

void set(const gstBasicTransformMatrix& mat) ;
(makes matrix userDefined).

void set(int i ,
int j ,
double value) ;
Set row "i" and column "j" of matrix to "value" (makes matrix userDefined).

void setCenter(const gstPoint& newCenter) ;
Set x,y,z components of center.

void setCenter(double argX ,
double argY ,
double argZ) ;
Set x,y,z components of center.

void setQuick(int i ,
int j ,
double value) ;
For internal use.

void setRotate(const gstVector& v ,double a) ;
DEPRECATED: Name changed for consistency.

void setRotation(const gstVector& axis ,double angle) ;
Set rotation using vector/axis angle approach. Axis is in radians.

void setRotationMatrix(const double r [3] [3]) ;
Set rotation matrix to user defined 3x3 matrix.

void setRotationMatrix(double r00 ,
double r01 ,
double r02 ,
double r10 ,

double r11 ,
double r12 ,
double r20 ,
double r21 ,
double r22) ;

Set rotation matrix (row 0 : columns 0-2, row 1 : columns 0-2, row 2 : columns 0-2).

void setScale(const gstPoint& newScale) ;
Set x,y,z components of scale.

void setScale(double argX ,
double argY ,
double argZ) ;
Set x,y,z components of scale.

void setTranslate(const gstPoint& newTranslation) ;
DEPRECATED: Name changed for consistency.

void setTranslate(double argX ,
double argY ,
double argZ) ;
DEPRECATED: Name changed for consistency.

void setTranslation(const gstPoint& newTranslation) ;
Set x,y,z components of translate.

void setTranslation(double argX ,
double argY ,
double argZ) ;
Set x,y,z components of translate.

gstPoint toLocal(const gstPoint& p) ;
Transform point in parent reference frame to local reference frame.

gstVector toLocal(const gstVector& v) ;
Transform vector in parent reference frame to local reference frame. Preserves vector length.

void translate(const gstPoint& newTranslation) ;
Add "newTranslation" to existing translation.

void translate(double argX ,
double argY ,
double argZ) ;
Add new x,y,z components to exisiting translation.

void transpose() ;
Set value of matrix to its transpose.

void update() const;
For internal use.

void updateInverse() ;
For internal use.

static void setSignalResetError(bool signal) ;

A user defined gstTransformMatrix will be reset to identity if a "convenience function" like "translate" or "rotate" is called on it. By default this reset triggers a GHOST error. Use setSignalResetError to turn this error reporting on/off.

class gstTransformMatrix::Proxy

Summary

#include “gstTransformMatrix.h”
class gstTransformMatrix::Proxy ;

Public Operators

__forceinline Proxy&	operator=(const Proxy& proxy) ;
__forceinline Proxy&	operator=(const double value) ;
__forceinline	operator double() const;
__forceinline Proxy&	operator*=(const double value) ;
__forceinline Proxy&	operator+=(const double value) ;
__forceinline Proxy&	operator-=(const double value) ;
__forceinline Proxy&	operator/=(const double value) ;

class gstTransformMatrix::Proxy1D

Summary

#include “gstTransformMatrix.h”
class gstTransformMatrix::Proxy1D ;

Public Operators

__forceinline Proxy1D&	operator=(const Proxy1D& proxy) ;
__forceinline Proxy1D&	operator=(const gstPoint value) ;
__forceinline	operator gstPoint() const;
__forceinline const Proxy	operator[](const int j) const;
__forceinline Proxy	operator[](const int j) ;

Function operator*

Summary #include "gstTransformMatrix.h"

```
__forceinline gstPoint operator*(const gstPoint& pt ,const
gstTransformMatrix& mat ) ;
```

Function operator*

Summary #include "gstTransformMatrix.h"

```
__forceinline gstPoint operator*(const gstTransformMatrix& mat ,const
gstPoint& pt ) ;
```

Function operator*

Summary #include "gstTransformMatrix.h"
```
__forceinline gstVector operator*(const gstTransformMatrix& mat ,const
gstVector& vec ) ;
```

Function operator*

Summary

#include "gstTransformMatrix.h"

```
__forceinline gstVector operator*(const gstVector& vec ,const
gstTransformMatrix& mat ) ;
```

Function operator<<

Summary #include "gstTransformMatrix.h"

```
ostream& operator<<(ostream& os ,const gstTransformMatrix& mat ) ;
```

Function operator<<

Summary #include "gstTransformMatrix.h"
```
__forceinline ostream& operator<<(ostream& os ,const
gstTransformMatrix::Proxy& e ) ;
```

class gstBoundedHapticObj

Summary

#include “gstBoundedHapticObj.h”
class gstBoundedHapticObj : public gstTransform;

Description

Base class for haptic objects that utilize a bounding volume.

Public constructors

virtual ~gstBoundedHapticObj() ;
Destructor.

Public Members

gstBoundedHapticObj* AsBoundedHapticObj() ;
Downcast function.

gstBoolean boundByBox(const gstPoint& center ,
const double xlen ,
const double ylen ,
const double zlen) ;
Utility method to bound the haptic object by a box.

gstBoolean boundBySphere(const gstPoint& center ,const double radius) ;
Utility method to bound the haptic object by a sphere.

virtual gstBoundingVolume* getBoundingVolume() const;
Get the gstBoundingVolume instance currently associated with the haptic object.

virtual double getDistanceFromBoundingVolumeWC(const gstPoint& point) ;
This method will return the closest distance from the input point to the bounding volume.

gstBoolean getNeedsUpdate() ;
Does the bounding volume need updating.

virtual gstType getTypeId() const;
Virtual form of getClassTypeId.

virtual void invalidateCumTransf() ;
Invalidate the cumulative transform for the object This will trigger the recalculation of the bounding sphere in world coordinates.

virtual gstBoolean isOfType(gstType type) const;
Virtual form of staticIsOfType.

virtual void setBoundingVolume(gstBoundingVolume* boundingObj) ;
Assign a gstBoundingVolume instance to the haptic object.

void setNeedsUpdate(gstBoolean t_or_f) ;
Set the state of the bounding volume w.r.t. Updating.

virtual gstBoolean withinBoundingVolume(const gstPoint& start ,const gstPoint& end) ;
Returns TRUE or FALSE depending on if the sphere intersects the bounding volume.

static gstType getClassTypeId() ;

Static: get type id of this class.

static gstBoolean staticIsOfType(gstType type) ;
Return TRUE if class is of the given type or is derived from that type.

Protected constructors

gstBoundedHapticObj() ;
This class is intended as a base class only, the constructors are protected so that instances can not be created.

gstBoundedHapticObj(const gstBoundedHapticObj& origBoundedObj) ;
Copy Ctor.

gstBoundedHapticObj(const gstBoundedHapticObj* origBoundedObj) ;

class gstBoundingBox

Summary

#include “gstBoundingBox.h”
class gstBoundingBox ;

Description

Represents a box aligned with the x, y, and z axis which contains a volume of space.

Enums

enum **returnValues**
RV_LEFT
RV_MIDDLE
RV_RIGHT

Public constructors

gstBoundingBox() ;
Front Right Top corner.

gstBoundingBox(vector<gstPoint>& v) ;
Ctor which takes a collection of points and calculates the bounding box.

gstBoundingBox(const gstPoint& blb ,const gstPoint& frt) ;

virtual ~gstBoundingBox() ;

Public Members

gstPoint center() ;
Returns the position of the center of gstBoundingBox.

gstBoundingSphere getBoundingSphere() ;
Returns gstSimpleSphere centered at center of gstBoundingBox and with radius such that gstBoundingBox is just enclosed within the sphere.

gstBoundingBox getLLBBox() ;
Returns new gstBoundingBox set to Left Lower Back quadrant of 'this' gstBoundingBox.

gstBoundingBox getLLFBox() ;
Returns new gstBoundingBox set to Left Lower Front quadrant of 'this' gstBoundingBox.

gstBoundingBox getLUBBox() ;
Returns new gstBoundingBox set to Left Upper Back quadrant of 'this' gstBoundingBox.

gstBoundingBox getLUFBox() ;
Returns new gstBoundingBox set to Left Upper Front quadrant of 'this' gstBoundingBox.

gstPoint getMaxPt() const;
Returns maximum position (right upper front corner).

gstPoint getMinPt() const;
Returns minimum position (left lower back corner).

gstBoundingBox getRLBBox() ;
Returns new gstBoundingBox set to Right Lower Back quadrant of 'this' gstBoundingBox.

gstBoundingBox getRLFBox() ;

Returns new gstBoundingBox set to Right Lower Front quadrant of 'this' gstBoundingBox.

gstBoundingBox getRUBBox() ;
Returns new gstBoundingBox set to Right Upper Back quadrant of 'this' gstBoundingBox.

gstBoundingBox getRUFBox() ;
Returns new gstBoundingBox set to Right Upper Front quadrant of 'this' gstBoundingBox.

gstBoolean inside(gstPoint& pt) ;
Tests if pt is inside of box. If so, TRUE is returned. Otherwise, FALSE is returned.

virtual gstLineIntersectionInfo::IntersectionType intersect(gstLineSegment& lineSegment ,gstLineIntersectionInfoFirstTwo_Param& intersectionInfo) ;
Intersects gstLineSegment with spatial object and returns gstIntersection::intersectionType which can be; in, out, inOut, none_inside, or none_outside. 'in' specifies an intersection into the object, 'out' specifies an intersection leaving the object, 'inOut' specifies intersections entering and leaving the object, 'none_insdie' specifies that the object is not intersected and the line segment is within the object, and 'none_outside' specifies that there is no intersection and the line segment is outside of the object.

virtual gstLineIntersectionInfo::IntersectionType intersect(gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_Param& intersectionInfo) ;
Intersects gstLineSegment with spatial object and returns gstIntersection::intersectionType which can be; in, out, inOut, none_inside, or none_outside. 'in' specifies an intersection into the object, 'out' specifies an intersection leaving the object, 'inOut' specifies intersections entering and leaving the object, 'none_insdie' specifies that the object is not intersected and the line segment is within the object, and 'none_outside' specifies that there is no intersection and the line segment is outside of the object.

virtual gstLineIntersectionInfo::IntersectionType intersect(gstRay& ray ,gstLineIntersectionInfoFirstTwo_Param& intersectionInfo) ;
Intersects gstRay with spatial object and returns gstIntersection::intersectionType which can be; in, out, inOut, or none. 'in' specifies an intersection into the object, 'out' specifies an intersection leaving the object, 'inOut' specifies intersections entering and leaving the object, and 'none' specifies that the object is not intersected.

virtual gstLineIntersectionInfo::IntersectionType intersect(gstRay& ray ,gstLineIntersectionInfoFirst_Param& intersectionInfo) ;
Intersects gstRay with spatial object and returns gstIntersection::intersectionType which can be; in, out, inOut, or none. 'in' specifies an intersection into the object, 'out' specifies an intersection leaving the object, 'inOut' specifies intersections entering and leaving the object, and 'none' specifies that the object is not intersected.

gstPoint leftLowerBackCorner() const;
Returns gstPoint coincident with the left lower back corner of this gstBoundingBox.

gstPoint leftLowerFrontCorner() ;
Returns gstPoint coincident with the left lower front corner of this gstBoundingBox.

gstPoint leftUpperBackCorner() ;
Returns gstPoint coincident with the left upper back corner of this gstBoundingBox.

gstPoint leftUpperFrontCorner() ;
Returns gstPoint coincident with the left upper front corner of this gstBoundingBox.

gstPoint rightLowerBackCorner() ;
Returns gstPoint coincident with the right lower back corner of this gstBoundingBox.

gstPoint rightLowerFrontCorner() ;
Returns gstPoint coincident with the right lower front corner of this gstBoundingBox.

gstPoint rightUpperBackCorner() ;
Returns gstPoint coincident with the right upper back corner of this gstBoundingBox.

gstPoint rightUpperFrontCorner() const;
Returns gstPoint coincident with the right upper front corner of this gstBoundingBox.

void setMaxPt(gstPoint newMaxPt) ;
Sets maximum position (right upper front corner) to newMaxPt.

void setMinPt(gstPoint newMinPt) ;
Sets minimum position (left lower back corner) to newMinPt.

static gstBoundingBox getLargestBoundingBox() ;
Returns a gstBoundingBox with DBL_MAX side lengths.

class gstBoundingCube

Summary

#include “gstBoundingCube.h”
class gstBoundingCube ;

Description

Represents a cube aligned with the x, y, and z axis which contains a volume of space.

Enums

enum **returnValues**
RV_LEFT
RV_MIDDLE
RV_RIGHT

Public constructors

gstBoundingCube(const gstPoint& _center ,const double _sideLength) ;

gstBoundingCube(const gstPoint& minPt ,const gstPoint& maxPt) ;
This constructor creates valid gstCube for minPt and maxPt of a valid cube. Otherwise the resulting gstBoundingCube will be of undefined side length.

virtual ~gstBoundingCube() ;

Public Members

gstPoint backCenter() const;
Returns gstPoint coincident with the center of the back side of this cube.

gstPoint bottomCenter() const;
Returns gstPoint coincident with the center of the bottom side of this cube.

gstPoint frontCenter() const;
Returns gstPoint coincident with the center of the front side of this cube.

gstBoundingSphere getBoundingSphere() const;
Returns gstBoundingSphere centered at center of gstBoundingBox and with radius such that gstBoundingBox is just enclosed within the sphere.

gstPoint getCenter() const;
Returns the position of the center of gstBoundingBox.

gstBoundingCube getLLBCube() const;
Returns new gstBoundingBox set to Left Lower Back quadrant of 'this' gstBoundingBox.

gstBoundingCube getLLFCube() const;
Returns new gstBoundingBox set to Left Lower Front quadrant of 'this' gstBoundingBox.

gstBoundingCube getLUBCube() const;
Returns new gstBoundingBox set to Left Upper Back quadrant of 'this' gstBoundingBox.

gstBoundingCube getLUFCube() const;
Returns new gstBoundingBox set to Left Upper Front quadrant of 'this' gstBoundingBox.

gstPoint getMaxPoint() const;
Returns maximum position (right upper front corner).

gstPoint getMinPoint() const;
Returns minimum position (left lower back corner).

gstBoundingCube getRLBCube() const;
Returns new gstBoundingBox set to Right Lower Back quadrant of 'this' gstBoundingBox.

gstBoundingCube getRLFCube() const;
Returns new gstBoundingBox set to Right Lower Front quadrant of 'this' gstBoundingBox.

gstBoundingCube getRUBCube() const;
Returns new gstBoundingBox set to Right Upper Back quadrant of 'this' gstBoundingBox.

gstBoundingCube getRUFCube() const;
Returns new gstBoundingBox set to Right Upper Front quadrant of 'this' gstBoundingBox.

double getSideLength() const;
Returns side length of cube.

gstBoolean inside(const gstPoint& pt) const;
Tests if pt is inside of box. If so, TRUE is returned. Otherwise, FALSE is returned.

virtual gstObjectIntersectionInfo::IntersectionType intersect(const gstBoundingSphere& sphere) const;
Intersects gstBoundingSphere with spatial object and returns gstIntersection::intersectionType which can be; enclosed, enclosing, overlapping, or none. 'enclosed' specifies that the sphere is enclosed within the object, 'enclosing' specifies that the sphere encloses the spatial object, 'overlapping' specifies that the sphere and the spatial object overlap eachother, and 'none' specifies that the sphere and spatial object occupy separate space.

virtual gstLineIntersectionInfo::IntersectionType intersect(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirstTwo_Param& intersectionInfo) const;
Intersects gstLineSegment with spatial object and returns gstIntersection::intersectionType which can be; in, out, inOut, none_inside, or none_outside. 'in' specifies an intersection into the object, 'out' specifies an intersection leaving the object, 'inOut' specifies intersections entering and leaving the object, 'none_insdie' specifies that the object is not intersected and the line segment is within the object, and 'none_outside' specifies that there is no intersection and the line segment is outside of the object.

virtual gstLineIntersectionInfo::IntersectionType intersect(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_Param& intersectionInfo) const;
Intersects gstLineSegment with spatial object and returns gstIntersection::intersectionType which can be; in, out, inOut, none_inside, or none_outside. 'in' specifies an intersection into the object, 'out' specifies an intersection leaving the object, 'inOut' specifies intersections entering and leaving the object, 'none_insdie' specifies that the object is not intersected and the line segment is within the object, and 'none_outside' specifies that there is no intersection and the line segment is outside of the object.

virtual gstLineIntersectionInfo::IntersectionType intersect(const gstRay& ray ,gstLineIntersectionInfoFirstTwo_Param& intersectionInfo) const;
Intersects gstRay with spatial object and returns gstIntersection::intersectionType which can be; in, out, inOut, or none. 'in' specifies an intersection into the object, 'out' specifies an intersection leaving the object, 'inOut' specifies intersections entering and leaving the object, and 'none' specifies that the object is not intersected.

virtual gstLineIntersectionInfo::IntersectionType intersect(const gstRay& ray ,gstLineIntersectionInfoFirst_Param& intersectionInfo) const;
Intersects gstRay with spatial object and returns gstIntersection::intersectionType which can be; in, out, inOut, or none. 'in' specifies an intersection into the object, 'out' specifies an intersection leaving the object, 'inOut' specifies intersections entering and leaving the object, and 'none' specifies that the object is not intersected.

gstPoint leftCenter() const;
Returns gstPoint coincident with the center of the left side of this cube.

gstPoint leftLowerBackCorner() const;
Returns gstPoint coincident with the left lower back corner of this gstBoundingBox.

gstPoint leftLowerFrontCorner() const;
Returns gstPoint coincident with the left lower front corner of this gstBoundingBox.

gstPoint leftUpperBackCorner() const;
Returns gstPoint coincident with the left upper back corner of this gstBoundingBox.

gstPoint leftUpperFrontCorner() const;
Returns gstPoint coincident with the left upper front corner of this gstBoundingBox.

gstPoint rightCenter() const;
Returns gstPoint coincident with the center of the right side of this cube.

gstPoint rightLowerBackCorner() const;
Returns gstPoint coincident with the right lower back corner of this gstBoundingBox.

gstPoint rightLowerFrontCorner() const;
Returns gstPoint coincident with the right lower front corner of this gstBoundingBox.

gstPoint rightUpperBackCorner() const;
Returns gstPoint coincident with the right upper back corner of this gstBoundingBox.

gstPoint rightUpperFrontCorner() const;
Returns gstPoint coincident with the right upper front corner of this gstBoundingBox.

void setCenter(const gstPoint& centerParam) ;
Sets the center of the cube at centerParam.

void setSideLength(const double sideLengthParam) ;
Sets the cube side length to sideLengthParam.

gstPoint topCenter() const;
Returns gstPoint coincident with the center of the top of this cube.

class gstBoundingSphere

Summary

#include “gstBoundingSphere.h”
class gstBoundingSphere ;

Description

Represents a bounding sphere.

Public constructors

gstBoundingSphere(const gstPoint& _center ,const double _radius) ;

virtual ~gstBoundingSphere() ;

Public Members

gstPoint getCenter() const;
Returns the position of the center of gstBoundingSphere.

double getRadius() const;
Returns the radius of gstBoundingSphere.

gstBoolean inside(const gstPoint& pt) const;
Tests if pt is inside gstSimpleSphere. If |sphereCenter-pt| <= sphereRadius, then TRUE is returned. Otherwise, FALSE is returned.

gstBoolean intersectP(const gstPoint& start_pt ,const gstPoint& end_pt) ;
This method will return TRUE if the line defined by the input points intersects the sphere.

void setCenter(const gstPoint& _center) ;
Sets the position of the center of gstBoundingSphere to _center.

void setRadius(const double _radius) ;
Sets the radius of gstBoundingSphere to _radius.

static gstBoundingSphere getLargestBoundingSphere() ;
Returns a gstBoundingSphere with DBL_MAX radius.

class gstBoundingVolume

Summary

#include “gstBoundingVolume.h”
class gstBoundingVolume ;

Description

GHOST Class to inherit from to define bounding volumes An instance of this class is pointed to by the gstBoundedHapticObj.

Public constructors

virtual ~gstBoundingVolume() ;
Destructor.

Public Members

virtual gstBoundingVolume* Clone() const = 0;
Cloning function.

virtual gstTransformedBoundingBox* asBox() ;

virtual gstTransformedBoundingSphere* asSphere() ;
Method to downcast a pointer of this type to a specific bounding volume class.

virtual gstPoint getCenter(int localOrWC = GST_LOCAL_FRAME) const = 0;
Virtual function for getting the center of the bounding volume.

virtual double getDistanceWC(const gstPoint& point) = 0;
This method will return the smallest distance between the input point and the bounding volume in world coordinates;

gstBoolean getNeedsUpdate() ;
Internal method Accessor for getting the update state of this bounding volume.

gstBoundedHapticObj* getOwnerNode() const;
Internal method Accessor for getting the owner node of this bounding volume.

virtual void setCenter(const gstPoint& center) = 0;
Virtual function for setting the center of the bounding volume.

void setNeedsUpdate(gstBoolean t_or_f) ;
Internal method Accessor for setting the update state of this bounding volume.

void setOwnerNode(gstBoundedHapticObj* owner) ;
Internal method Accessor for setting the owner node of this bounding volume.

virtual gstBoolean testLineIntWC(const gstPoint& startPt ,const gstPoint& endPt) = 0;
Predicate method as to whether a sphere intersects the volume.

virtual gstBoolean update(gstTransform* owner) = 0;
Updates the position of the bounding volume object.

gstBoolean withinBoundingVolume(const gstPoint& startPt ,const gstPoint& endPt) ;
Internal Method
This method will determine if the line segment specified by startPt and endPt

```
intersects the bounding volume.
```

Protected constructors

gstBoundingVolume() ;

This class is intended as a base class only, the constructors are protected so that instances can not be created.

gstBoundingVolume(const gstBoundingVolume& origBounder) ;

gstBoundingVolume(const gstBoundingVolume* origBounder) ;

class gstTransformedBoundingBox

Summary

#include “gstTransformedBoundingBox.h”
class gstTransformedBoundingBox : public gstBoundingVolume;

Description

A class that implements a bounding box bounding volume implementation for use by the PHANToM. This implementation keeps a representation of the bounding volume in both local and world coordinates. The world coordinate representation is kept up to date automatically and is used to optimize the checking of whether the PHANToM is within the focus of the owning haptic object.

Public constructors

gstTransformedBoundingBox() ;
Constructors.

gstTransformedBoundingBox(const gstTransformedBoundingBox& origBox) ;
Constructor.

gstTransformedBoundingBox(const gstTransformedBoundingBox* origBox) ;
Constructor.

gstTransformedBoundingBox(const gstPoint& center ,
const double xlen ,
const double ylen ,
const double zlen) ;
Constructor.

~gstTransformedBoundingBox() ;
Destructors.

Public Members

virtual gstBoundingVolume* Clone() const;
Cloning function.

gstTransformedBoundingBox* CloneBoundBox() const;
Cloning function.

virtual gstTransformedBoundingBox* asBox() ;
Get the bounding volume of the correct type.

gstBoundingBox* getBox(int localOrWC) const;
Get the normal bounding box representation.

gstBoundingBox* getBoxLocal() const;
Internal method Get the local coordinate representation of this bounding box.

gstBoundingBox* getBoxWC() const;
Internal method Get the world coordinate representation of this bounding box.

virtual gstPoint getCenter(int localOrWC = GST_LOCAL_FRAME) const;
Accessor to set the center of the bounding box in local coordinates.

virtual double getDistanceWC(const gstPoint& point) ;
This method will return the closest distance between the input point and the bounding volume.

double getXlength(int localOrWC = GST_LOCAL_FRAME) const;
Accessor to set the x-axis length of the bounding box in local coordinates.

double getYlength(int localOrWC = GST_LOCAL_FRAME) const;
Accessor to set the y-axis length of the bounding box in local coordinates.

double getZlength(int localOrWC = GST_LOCAL_FRAME) const;
Accessor to set the z-axis length of the bounding box in local coordinates.

void setBoxLocal(gstBoundingBox* newBox) ;
Internal method Set the local coordinate representation of this bounding box.

void setBoxWC(gstBoundingBox* newBox) ;
Internal method Set the world coordinate representation of this bounding box.

virtual void setCenter(const gstPoint& center) ;
Accessor to set the center of the bounding box in local coordinates.

void setXlength(const double xlength) ;
Accessor to set the x-axis length of the bounding box in local coordinates.

void setYlength(const double ylength) ;
Accessor to set the y-axis length of the bounding box in local coordinates.

void setZlength(const double zlength) ;
Accessor to set the z-axis length of the bounding box in local coordinates.

virtual gstBoolean testLineIntWC(const gstPoint& startPt ,const gstPoint& endPt) ;
This method will test whether the input line intersects the bounding box.

virtual gstBoolean update(gstTransform* owner) ;
This method will update the WC version of the bounding volume based on the transform of the owner passed in.

Protected members

void convertLengthsToPoints(const gstPoint& center ,
const double length ,
const double width ,
const double height ,
gstPoint& blb ,
gstPoint& frt) const;

void convertPointsToLengths(const gstPoint& blb ,
const gstPoint& frt ,
gstPoint& center ,
double& length ,
double& width ,
double& height) const;

Helper methods.

gstBoolean transformBoxToWorld(gstTransformMatrix& cumMat) ;
This method will take the 3D axis-aligned box, apply the cumulative matrix's rotation, translation and scale, and determine a new axis-aligned box that will minimally contain the originally transformed box It is used for sphereIntersectionP. Algorithm courtesy of "Graphics Gems, Transforming Axis-Aligned Bounding Boxes".

class gstTransformedBoundingSphere

Summary

#include “gstTransformedBoundingSphere.h”
class gstTransformedBoundingSphere : public gstBoundingVolume;

Description

A class that implements a bounding sphere bounding volume implementation for use by the PHANToM. This implementation keeps a representation of the bounding volume in both local and world coordinates. The world coordinate representation is kept up to date automatically and is used to optimize the checking of whether the PHANToM is within the focus of the owning haptic object.

Public constructors

gstTransformedBoundingSphere() ;
Constructors.

gstTransformedBoundingSphere(const gstTransformedBoundingSphere& origSphere) ;
Constructor.

gstTransformedBoundingSphere(const gstTransformedBoundingSphere* origSphere) ;
Constructor.

gstTransformedBoundingSphere(const gstPoint& center ,double radius) ;
Constructor.

~gstTransformedBoundingSphere() ;
Destructors.

Public Members

virtual gstBoundingVolume* Clone() const;
Cloning function.

gstTransformedBoundingSphere* CloneBoundSphere() const;
Cloning function.

virtual gstTransformedBoundingSphere* asSphere() ;
Get the bounding volume of the correct type.

virtual gstPoint getCenter(int localOrWC = GST_LOCAL_FRAME) const;
Accessor for getting the center of the bounding volume.

virtual double getDistanceWC(const gstPoint& point) ;
A method to get to closest distance from point to the bounding sphere.

double getRadius(int localOrWC = GST_LOCAL_FRAME) const;
Accessor for setting the radius of the bounding volume.

gstBoundingSphere* getSphere(int localOrWC) const;
Get the normal bounding sphere representation of the bounding volume.

gstBoundingSphere* getSphereLocal() const;
Get the local coordinate representation of the bounding volume.

gstBoundingSphere* getSphereWC() const;
Get the world coordinate representation of the bounding volume.

virtual void setCenter(const gstPoint& center) ;
Accessor for setting the center of the bounding volume in local coordinates.

void setRadius(const double radius) ;
Accessor for setting the radius of the bounding volume in local coordinates.

void setSphereLocal(gstBoundingSphere* newSphere) ;
Set the local coordinate representation of the bounding volume.

void setSphereWC(gstBoundingSphere* newSphere) ;
Set the world coordinate representation of the bounding volume.

virtual gstBoolean testLineIntWC(const gstPoint& startPt ,const gstPoint& endPt) ;
A method to determine if the line segment specified by startPt and endPt intersects the bounding volume.

virtual gstBoolean update(gstTransform* owner) ;
A method to update the world coordinate representation of the bounding sphere based on the local representation and translations, rotations and scales from the world coordinate system.

Function getBoundingRadius

Summary

#include "gstBoundedHapticObj.h"

```
double getBoundingRadius(gstBoundedHapticObj* bounded ) ;
```

Description

Helper function for getting the radius of the bounding sphere instance associated with the haptic object.

Function setBoundingCenter

Summary #include "gstBoundedHapticObj.h"

```
void setBoundingCenter(gstBoundedHapticObj* bounded ,const gstPoint& radius
) ;
```

Description Helper function for setting the center of the bounding volume instance associated to the haptic object.

Function setBoundingDimensions

Summary

#include "gstBoundedHapticObj.h"

```
void setBoundingDimensions(gstBoundedHapticObj* bounded ,
                           double xlen ,
                           double ylen ,
                           double zlen ) ;
```

Description Helper function for setting the x-axis length of the bounding box instance associated with the haptic object.

Function setBoundingRadius

Summary #include "gstBoundedHapticObj.h"
`void setBoundingRadius(gstBoundedHapticObj* bounded ,double radius ) ;`

Description Helper function for setting the radius of the bounding sphere instance associated with the haptic object.

Chapter Supporting Structures

struct gstCollisionInfoStruct

Summary

#include “gstCollisionInfo.h”
struct gstCollisionInfoStruct ;

Description

Specifics about a single collision between the PHANToM and a shape.

Public Data

double	Kobj
gstPoint	SCP
gstVector	SCPnormal
double	dynamicFriction
gstShape*	obj
double	staticFriction
double	surfaceDamping

struct gstDimensionsStruct

Summary

#include “gstPHANToMInfo.h”
struct gstDimensionsStruct ;

Description

Workspace dimensions.

Public Data

int xMin,	xMax
int yMin,	yMax
int zMin,	zMax

class gstEventStack

Summary

#include “gstEventStack.h”
class gstEventStack ;

Description

Stack for storing gstTransform events.

Public constructors

gstEventStack() ;
Constructor.

~gstEventStack() ;
Destructor.

Public Members

void clear() ;
Clear event stack.

gstBoolean pop(gstEvent& nextEvent) ;
Pop event from event stack.

gstBoolean push(const gstEvent& newEvent) ;
Push event onto event stack.

Protected Data

int numEvents, eventIndexTop, eventIndexBottom
gstEvent eventStack [EVENT_STACK_SIZE]

struct gstPHANToMInfoStruct

Summary

#include “gstPHANToMInfo.h”
struct gstPHANToMInfoStruct ;

Description

Information about a particular PHANToM model.

Public Data

gstDimensionsStruct	cubeWorkspace - Maximizes reachable extents for a cube workspace.
gstBoolean	hasGimbal - Properties about the phantom hardware.
gstBoolean	is6DOF - Determines whether this phantom has 3DOF or 6DOF capabilities.
gstBoolean	isDesktop
gstDimensionsStruct	maxUsableWorkspace - Maximizes reachable extents for a rectangular workspace.
gstDimensionsStruct	maxWorkspace - Workspace dimensions based on the phantom model type or registry settings some extents may not be reachable.
int	tableTopOffset - Offset in the y direction from the reset position.

Function gstDisablePhantom

Summary #include "gstDeviceIO.h"

```
int gstDisablePhantom(int id ) ;
```

Description Disables the phantom whose id is given.

Function gstDisablePhantomForces

Summary #include "gstDeviceIO.h"
```
int gstDisablePhantomForces(int id ) ;
```

Description Disables the amplifiers so the Phantom stops sending forces.

Function gstEnablePhantomForces

Summary

#include "gstDeviceIO.h"

```
int gstEnablePhantomForces(int id ) ;
```

Description

Enables the amplifiers so that forces can be sent to the phantom. (typically done once at the beginning).

Function gstEncodersToTransform

Summary

#include "gstDeviceIO.h"

```
int gstEncodersToTransform(int id ,
                           long encoders [ ] ,
                           gstTransformMatrix& mat ) ;
```

Description

Performs Phantom kinematics to convert encoder values to a stylus transform. This does not affect the phantom's intenral picture of itself - if you've changed the encodrs so that you get a different answer, you may get weird results.

Function gstGetJointAngles

Summary

#include "gstDeviceIO.h"

```
int gstGetJointAngles(int id ,gstVector& theta ) ;
```

Description

Gets the phantom joint angles. Order: (left(+); up(+); out(+)).

Function gstGetPhantomError

Summary

#include "gstDeviceIO.h"

```
int gstGetPhantomError(int id ) ;
```

Description

Return a device fault (but not other phantom error states) It makes more sense to check the return state of updatePhantom to find out what the current error state is.

Function gstGetPhantomInfo

Summary

#include "gstDeviceIO.h"

```
int gstGetPhantomInfo(int id ,gstPHANToMInfoStruct& pInfo ) ;
```

Description

Gets the gstPhantomInfoStruct, which contains various data on the current Phantom.

Function gstGetPhantomMaxStiffness

Summary

#include "gstDeviceIO.h"

```
float gstGetPhantomMaxStiffness(int id ) ;
```

Description

Gets the MaxStiffness for the phantom whose ID is given. MaxStiffness should be used to scale all phantom stiffnesses, if you plan to use your code with multiple phantom models.

Function gstGetPhantomPosition

Summary #include "gstDeviceIO.h"

```
int gstGetPhantomPosition(int id ,gstPoint& pos ) ;
```

Description Gets the Phantom position in cartesian space.

Function gstGetPhantomTemperature

Summary

#include "gstDeviceIO.h"

```
int gstGetPhantomTemperature(int id ,float temp [ ] ) ;
```

Description

Gets the Phantom temperature (according to internal temp model.) returns 6 values (3 will be zero for non-6dofs).

Function gstGetPhantomUpdateRate

Summary #include "gstDeviceIO.h"

```
float gstGetPhantomUpdateRate(int id ) ;
```

Description Gets the instantaneous phantom update rate.

Function gstGetPhantomVelocity

Summary

#include "gstDeviceIO.h"

```
int gstGetPhantomVelocity(int id ,gstVector& vel ) ;
```

Description

Gets the phantom velocity in cartesian space. Will only work if the update rate is greater than 900 Hz. Otherwise, returns last known "good" velocity.

Function gstGetRawEncoderValues

Summary

#include "gstDeviceIO.h"

```
int gstGetRawEncoderValues(int id ,long encoders [ ] ) ;
```

Description

Gets the raw encoder values (order: the 3 base encoders, then the 3 gimbal encoders).

Function gstGetStylusJointAngles

Summary #include "gstDeviceIO.h"

```
int gstGetStylusJointAngles(int id ,gstVector& theta ) ;
```

Description Gets the phantom stylus joint angles. Order: (right(+), up(+), right(+)).

Function gstGetStylusMatrix

Summary

#include "gstDeviceIO.h"

```
int gstGetStylusMatrix(int id ,gstTransformMatrix& mat ) ;
```

Description

Gets the transform matrix representing the stylus position and orientation.

Function gstGetStylusSwitchState

Summary #include "gstDeviceIO.h"

```
int gstGetStylusSwitchState(int id ) ;
```

Description Gets the state of the stylus switch.

Function gstInitServoScheduler

Summary #include "gstDeviceIO.h"

```
int gstInitServoScheduler() ;
```

Description Intialization necessary to start a servo loop.

Function gstInitializePhantom

Summary

#include "gstDeviceIO.h"

```
int gstInitializePhantom(char* phantomName ) ;
```

Description

Initializes the Phantom - must call before you do anything else with the phantom, returns a phantom ID if successful, and one of the error codes above if not.

Function gstIsPhantomResetNeeded

Summary

#include "gstDeviceIO.h"

```
bool gstIsPhantomResetNeeded(int id ) ;
```

Description

Returns whether the given phantom id is a phantom type that requires a reset or not.

Function gstResetPhantomEncoders

Summary

#include "gstDeviceIO.h"

```
int gstResetPhantomEncoders(int id ) ;
```

Description

Sets the phantom encoders to zero. Should be used by all phantoms that require a reset.

Function gstSetPhantomForce

Summary

#include "gstDeviceIO.h"

```
int gstSetPhantomForce(int id ,const gstVector& force ) ;
```

Description

Sends a force to the phantom motors. Use this version for non-6dof.

Function gstSetPhantomForce

Summary

#include "gstDeviceIO.h"

```
int gstSetPhantomForce(int id ,
                       const gstVector& force ,
                       const gstVector& torque ) ;
```

Description

Sends a force to the Phantom motors. Use this version for 6dof. The force vector is cartesian forces, the torque vector is joint torques for the 6dof axes.

Function gstStartServoScheduling

Summary #include "gstDeviceIO.h"

```
int gstStartServoScheduling(gstServoSchedulerCallback pCallback ,void*
userData ) ;
```

Description Start the servo loop whose callback is given here.

Function gstStopServoScheduling

Summary #include "gstDeviceIO.h"

```
void gstStopServoScheduling() ;
```

Description Stop the currently running servo loop.

Function gstUpdatePhantom

Summary

#include "gstDeviceIO.h"

```
int gstUpdatePhantom(int id ) ;
```

Description

Updates the Phantom's internal picture of itself by reading encoders, calculating position, etc. Should call each servo loop (if creating own).

class gstEdge<gstTriPoly*, class __default_alloc_template< 1, 0>>

Summary

#include “gstEdge.h”
template <gstTriPoly*, class __default_alloc_template< 1, 0>>
class gstEdge ;

Description

Mesh element representing an edge of one or more polygonal faces with two gstVertex elements at it's end points (v1 and v2).

Public constructors

gstEdge(gstVertex* v1new ,
gstVertex* v2new ,
gstTriPoly* p1 = NULL,
gstTriPoly* p2 = NULL) ;

virtual ~gstEdge() ;

Public Members

gstLineSegment getLineSegment() const;
Returns a gstLineSegment with p1 and p2 of the line segment coincident to v1 and v2 of this edge.

int getNumIncidentPolys() const;
Returns number of polygons sharing this edge.

gstTriPoly* getP1() const;
Returns the first incident poly to this edge from an unordered list. If there are no incident polygons to this edge, then NULL is returned.

gstTriPoly* getP2() const;
Returns the second incident poly to this edge from an unordered list. If there are less than 2 incident polygons to this edge, then NULL is returned.

gstVertex* getV1() const;
Returns a pointer to gstVertex v1.

gstVertex* getV2() const;
Returns a pointer to gstVertex v2.

gstTriPolyPtrListConstIterator incidentPolysBegin() const;
Returns an iterator at the beginning position of a list of the incident polygons to this edge.

gstTriPolyPtrListConstIterator incidentPolysEnd() const;
Returns an iterator at the ending position of a list of the incident polygons to this edge.

gstBoolean isStranded() const;
Returns TRUE if the number of incident polygons is 0.

int project(const gstPoint& pt ,gstPoint* projectedPt = NULL) const;
Projects pt onto edge and returns projection in projectedPt. If projectedPt lies to the right of v1 then gstEdge::LEFT is returned. If projectedPt lies in between v1 and v2 then gstEdge::BETWEEN is returned. Otherwise, gstEdge::RIGHT is returned.

gstVertex* v1() const;
Returns a pointer to gstVertex v1.

gstVertex* v2() const;
Returns a pointer to gstVertex v2.

Protected members

void addIncidentPoly(gstTriPoly* poly) ;
Adds poly to the list of incident polygons for this edge.

int removeIncidentPoly(const gstTriPoly* polyToRemove) ;
Removes polyToRemove from list of incident polys to edge. If there are no more incident polys left then gstEdge::STRANDED is returned. If the poly wasn't incident to this then FALSE is returned. Otherwise, TRUE is returned.

class gstIncidentEdge

Summary

#include “gstIncidentEdge.h”
class gstIncidentEdge ;

Description

Type of edge used by gstTriPoly.

Enums

enum **ReturnValues_**
RV_IN
RV_OUT
RV_CLOCKWISE
RV_COUNTERCLOCKWISE

Public constructors

gstIncidentEdge(gstEdge* _edge = NULL,const int _direction = gstIncidentEdge::RV_IN) ;

Public Operators

gstBoolean operator<(const gstIncidentEdge& e) const;
Less than operator.

gstBoolean operator==(const gstIncidentEdge& e) const;
Equality test operator.

Public Members

int getDirection() const;
Declare STL routines used by GHOST to be exported symbols from the dll.

gstEdge* getEdge() const;

gstVertex* getOppositeVertex() const;
Returns pointer to gstVertex on opposite side of incident edge.

void setDirection(const int _direction) ;

void setEdge(gstEdge* _edge) ;

class gstLine

Summary

#include “gstLine.h”
class gstLine : public gstLineBase;

Description

Implements an infinite line passing through space.

Public constructors

gstLine(const gstPoint& _p1 ,const gstPoint& _p2) ;
Constructor defines line as passing through points p1 and p2. The parametric representation is then f(t) = (1-t)P1+tP2.

gstLine(const gstPoint& _p1 ,const gstVector& _v1) ;
Constructor defines line passing through p1 and directed along v1. The parametric form is defined as f(t)=(1-t)P1+t(P1+V1). Thus f(t)=p1 at t=0 and f(t)=p1+v1 at t=1.

virtual ~gstLine() ;
Line passes through p2.

Public Members

virtual gstVector direction() const;
GstVector directed along line in the direction of v1 or toward p2 depending on constructor used.

virtual gstPoint eval(const double t) const;
Evaluates parametric form of line at t and returns point at f(t).

virtual const gstPoint& getPointOnLine() const;
Returns a point on the line.

virtual const gstPoint& pointOnLine() const;
Returns a point on the line.

virtual gstVector unitDirection() const;
GstVector directed along line in the direction of v1 or toward p2 depending on constructor used.

class gstLineBase

Summary

#include “gstLineBase.h”
class gstLineBase ;

Description

Base class for all types of lines passing through space (ie. Ray, line segment...).

Public constructors

gstLineBase() ;
Line is directed along v1.

gstLineBase(gstPoint& _p1 ,gstPoint& _p2) ;
Constructor defines line as passing through points p1 and p2. The parametric representation is then f(t) = (1-t)P1+tP2.

gstLineBase(gstPoint& _p1 ,gstVector& _v1) ;
Constructor defines line passing through p1 and directed along v1. The parametric form is defined as f(t)=(1-t)P1+t(P1+V1). Thus f(t)=p1 at t=0 and f(t)=p1+v1 at t=1.

virtual ~gstLineBase() ;

Public Members

virtual gstVector direction() const = 0;
GstVector directed along line in the direction of v1 or toward p2 depending on constructor used.

virtual gstPoint eval(const double t) const = 0;
Evaluates parametric form of line at t and returns point at f(t).

virtual const gstPoint& getPointOnLine() const = 0;
Returns a point on the line.

virtual gstPoint project(const gstPoint& pt) const;
Projects a point onto the line and returns the projected point on the line.

virtual gstVector unitDirection() const;
GstVector directed along line in the direction of v1 or toward p2 depending on constructor used.

class gstLineSegment

Summary

#include “gstLineSegment.h”
class gstLineSegment : public gstLineBase;

Description

Implements an line segment in space defined by two endpoints, p1 and p2.

Public constructors

gstLineSegment(const gstLineSegment& lineSeg) ;

gstLineSegment(const gstLineSegment* lineSeg) ;
Line passes through p2.

gstLineSegment(const gstPoint& p1 ,const gstPoint& p2) ;
Constructor defines line segment from p1 to p2. The parametric representation is then f(t) = (1-t)P1+tP2.

gstLineSegment(const gstPoint& p1 ,const gstVector& v1) ;
Constructor defines line segment starting at p1 directed along v1 and ending at p1+v1. The parametric form is defined as f(t)=(1-t)P1+t(P1+V1). Thus f(t)=p1 at t=0 and f(t)=p1+v1 at t=1.

virtual ~gstLineSegment() ;

Public Members

virtual gstVector direction() const;
GstVector directed along line in the direction of v1 or toward p2 depending on constructor used.

virtual gstPoint eval(const double t) const;
Evaluates parametric form of line segment at t and returns point at f(t). P1 is returned for values of t<0 and p2 is returned for values of t > 1.

double eval(const gstPoint& pt) const;
Gives the parametric value t of the line segment for the projection of pt onto the line segment.

const gstPoint& getEndPoint() const;
Returns p2.

virtual const gstPoint& getPointOnLine() const;
Returns a point on the line.

const gstPoint& getStartPoint() const;
Returns p1.

double length() const;
Returns length of line segment.

gstPoint origin() const;
Returns p1.

double projectToParametric(const gstPoint& pt) const;
Projects pt onto the line segment and returns the parametric value of the projection.

void setEndPoint(const gstPoint& endPoint) ;
Sets p2 = endPoint.

void setStartPoint(const gstPoint& startPoint) ;
Sets p1 = startPoint.

virtual gstVector unitDirection() const;
GstVector directed along line in the direction of v1 or toward p2 depending on constructor used.

class gstPlane

Summary

#include “gstPlane.h”
class gstPlane : public gstSpatialObject;

Description

Represents a geometric plane.

Public constructors

gstPlane(const gstPlane& plane) ;
Constructor.

gstPlane(const gstPlane* plane) ;
Constructor.

gstPlane(const gstVector& normalArg ,const gstPoint& p) ;

gstPlane(const gstVector& normalArg ,double dArg) ;
Constructor.

gstPlane(const gstPoint& point1 ,
const gstPoint& point2 ,
const gstPoint& point3) ;
Constructor.

gstPlane(double a = 0. 0,
double b = 1. 0,
double c = 0. 0,
double d = 0. 0) ;
Constructor.

virtual ~gstPlane() ;

Public Operators

bool operator==(const gstPlane& rhs) const;
Comparison.

Public Members

double a() const;
Returns A coefficient from plane equation.

double b() const;
Returns B coefficient from plane equation.

double c() const;
Returns C coefficient from plane equation.

virtual gstSpatialObject* clone() const;

gstPlane* clonePlane() const;

double d() const;
Returns D coefficient from plane equation.

double error(const gstPoint& pt) const;
Returns ax + by + cz + d.

virtual gstType getTypeId() const;
Virtual form of getClassTypeId.

virtual gstLineIntersectionInfo::IntersectionType intersectFirstInOut_LS_P(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_Param& intersectionInfo) const;

virtual gstLineIntersectionInfo::IntersectionType intersectFirstInOut_Ray_P(const gstRay& ray ,gstLineIntersectionInfoFirst_Param& intersectionInfo) const;

virtual gstLineIntersectionInfo::IntersectionType intersectFirstIn_LS_P(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_Param& intersectionInfo) const;

virtual gstLineIntersectionInfo::IntersectionType intersectFirstOut_LS_P(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_Param& intersectionInfo) const;

int intersectPlane(const gstPlane& interPlane ,
gstRay& interLine ,
double tol = 0. 0000000001) const;
Find intersection between a given gstPlane and this one. The resulting intersection is returned as a gstRay, if one exists. Tol is acceptable distance for parallel planes to be considered as coincident. Return value is: 1 - unique intersection as a line, returned as a gstRay 0 - planes are coincident (no gstRay returned) -1 - planes don't intersect (no gstRay returned).

virtual gstBoolean isOfType(gstType type) const;
Virtual form of staticIsOfType.

virtual gstVector normal() const;
Returns normal vector of plane.

gstVector perpVec(const gstPoint& pt) const;
Returns vector perpendicular to point p.

gstPoint pointOfLineIntersection(const gstPoint& p1 ,const gstPoint& p2) const;
Given line passing through p1 and p2, intersects plane. Interection is set to point of intersection with line and TRUE is returned. If the line does not intersect, returns FALSE.

gstBoolean pointOfLineIntersection(const gstPoint& point1 ,
const gstPoint& point2 ,
gstPoint& intersection) const;
Given line through p1 and p2, returns TRUE if line intersects plane with point of intersection in intersection argument. FALSE otherwise.

gstBoolean pointOfSegmentIntersection(const gstPoint& point1 ,
const gstPoint& point2 ,
gstPoint& intersection) const;
Given endpoints of line segment, p1 and p2, returns TRUE if line segment intersects plane with point of intersection in intersection argument. The plane is considered one-sided, in the direction of the normal; if the segment is in the same direction as the normal, no intersection will be returned. Returns FALSE if there is no intersection.

gstPoint pointOnPlane() const;

Returns some point on the plane.

void printSelf(FILE* outf) const;
For internal use.

void printSelf2() const;
Print to stdout.

gstPoint project(const gstPoint& p ,double offsetFactor = 1. 00001) const;
Returns projection of point p onto plane.

gstPoint projectPoint(const gstPoint& p) const;
Returns projection of point p onto plane.

bool projectPointAlongVector(const gstPoint& p ,
const gstVector& v ,
gstPoint& resultPt) const;
Returns projection of point p onto plane
along given vector.

void setD(double _d) ;

void setPlane(const gstVector& newNormal ,const gstPoint& newIntersection) ;
Set plane to be located at position indicated by normal and point.

static gstType getClassTypeId() ;
Static: get type id of this class.

static gstBoolean staticIsOfType(gstType type) ;
Return TRUE if class is of the given type or is derived from that type.

Protected Data

double D
gstVector normalVector

class gstPoint

Summary

#include “gstPoint.h”
class gstPoint ;

Description

Represents a point in 3D space (x, y, z).

Public constructors

gstPoint() ;
Constructor.

gstPoint(const gstPoint& p) ;
Constructor.

gstPoint(const gstPoint* p) ;
Constructor.

gstPoint(double x ,
double y ,
double z) ;
Constructor.

Public Operators

gstPoint& operator=(const gstPoint& p) ;
Assignment operator.

gstPoint& operator=(const gstPoint* p) ;
Assignment operator.

operator double*() ;

gstBoolean operator!=(const gstPoint& p) const;
Inequality test operator.

gstPoint operator+(const gstPoint& p) const;
Addition operator.

gstPoint& operator+=(const gstPoint& p) ;
Accumulation (add and assign) operator.

gstPoint operator-() const;
Subtraction operato.r.

gstPoint operator-(const gstPoint& p) const;
Substraction operator.

gstPoint& operator-=(const gstPoint& p) ;
Subtract and assign operator.

gstBoolean operator<(const gstPoint& p) const;
Less than operator.

gstBoolean operator==(const gstPoint& p) const;

Equality test operator.

gstBoolean operator>(const gstPoint& p) const;
Greater than operator.

const double& operator[](int i) const;
Returns coords[i], allowing point to
be accessed and treated as an array.

double& operator[](int i) ;
Returns coords[i], allowing point to
be accessed and treated as an array.

Public Members

double distToOrigin() ;
Returns magnitude of distance of point to origin.

const double* getValue() const;
Returns pointer to dynamically allocated double array containing x,y,z coordinates.

void getValue(double& x ,
double& y ,
double& z) ;
Returns x,y,z coordinates.

void init(double x ,
double y ,
double z ,
double w) ;

void init(double x ,
double y ,
double z) ;
double coords[0], coords[1], coords[2];
Constructor.

gstBoolean isZero() const;
Check for zero operator.

void printSelf() ;
Print to stdout.

void setx(double x) ;
Set coords[0] coordinate.

void sety(double y) ;
Set coords[1] coordinate.

void setz(double z) ;

Set coords[2] coordinate.

double w() const;
Returns 4th homogeneous component of point.

gstBoolean withinEpsilon(const gstPoint& pt ,double epsilon = 0. 00000001) const;
Check if point is (nearly) the same.

double x() const;
Returns coords[0] coordinate.

double y() const;
Returns coords[1] coordinate.

double z() const;
Returns coords[2] coordinate.

Protected Data

double coords [4]

class gstPoint2D

Summary

#include “gstPoint2D.h”
class gstPoint2D ;

Description

Represents a point in 2D space (x, y).

Public constructors

gstPoint2D() ;
Constructor.

gstPoint2D(const gstPoint2D& p) ;
Constructor.

gstPoint2D(const gstPoint2D* p) ;
Constructor.

gstPoint2D(double u ,double v) ;
Constructor.

Public Operators

gstPoint2D& operator=(const gstPoint2D& p) ;
Assignment operator.

gstPoint2D& operator=(const gstPoint2D* p) ;
Assignment operator.

gstBoolean operator!=(const gstPoint2D& p) const;
Inequality test operator.

gstPoint2D& operator*=(double d) ;
Multiply and assign operator.

gstPoint2D operator+(const gstPoint2D& p) const;
Addition operator.

gstPoint2D& operator+=(const gstPoint2D& p) ;
Accumulation (add and assign) operator.

gstPoint2D operator-() const;
Subtraction operato.r.

gstPoint2D operator-(const gstPoint2D& p) const;
Substraction operator.

gstPoint2D& operator-=(const gstPoint2D& p) ;
Subtract and assign operator.

gstPoint2D& operator/=(double d) ;
Divide and assign operator.

gstBoolean operator<(const gstPoint2D& p) const;
Less than operator.

gstBoolean operator==(const gstPoint2D& p) const;
Equality test operator.

gstBoolean operator>(const gstPoint2D& p) const;
Greater than operator.

const double& operator[](int i) const;
Returns coords[i], allowing point to be accessed and treated as an array.

double& operator[](int i) ;
Returns coords[i], allowing point to be accessed and treated as an array.

Public Members

double distToOrigin() ;
Returns magnitude of distance of point to origin.

const double* getValue() const;
Returns pointer to dynamically allocated double array containing u, v coordinates.

void getValue(double& u ,double& v) ;
Returns x,y,z coordinates.

void init(double u ,double v) ;
Double coords[0], coords[1], Constructor.

gstBoolean isZero() const;
Check for zero operator.

void printSelf() ;
Print to stdout.

void setu(double u) ;
Set coords[0] coordinate.

void setv(double v) ;
Set coords[1] coordinate.

double u() const;
Returns coords[0] coordinate.

double v() const;
Returns coords[1] coordinate.

Protected Data

double coords [2]

class gstQuaternion

Summary

#include “gstQuaternions.h”
class gstQuaternion ;

Description

Implement quaternions (to represent rotations).

Public constructors

gstQuaternion() ;

gstQuaternion(double (* r) [3]) ;

gstQuaternion(double _s ,gstVector _v) ;

gstQuaternion(gstVector axis ,double ang) ;

Public Operators

gstQuaternion	operator*=(const gstQuaternion& q1) ;

Public Members

void	normalize() ;
void	scale(double s) ;
void	toAxisAngle(gstVector& axis ,double& radians) ;
void	toMatrix(double (* r) [3]) ;

Public Data

double	s
gstVector	v

class gstRay

Summary

#include “gstRay.h”
class gstRay : public gstLineBase;

Description

Implements a ray originating at p1 and directed along the vector v1. This ray is defined parametrically such that f(0.0)=p1 and f(1.0)=p1+v1.

Public constructors

gstRay() ;
Line passes through p2.

gstRay(const gstPoint& p1 ,const gstPoint& p2) ;
Constructor defines ray starting at p1 and passing through p2 at t=1. The parametric representation is then f(t) = (1-t)P1+tP2.

gstRay(const gstPoint& p1 ,const gstVector& v1) ;
Constructor defines ray starting at p1 directed along v1. The parametric form is defined as f(t)=(1-t)P1+t(P1+V1). Thus f(t)=p1 at t=0 and f(t)=p1+v1 at t=1.

virtual ~gstRay() ;

Public Operators

bool operator==(const gstRay& rhs) const;
Comparison.

Public Members

virtual gstVector direction() const;
GstVector directed along line in the direction of v1 or toward p2 depending on constructor used.

virtual gstPoint eval(const double t) const;
Evaluates parametric form of line segment at t and returns point at f(t). P1 is returned for values of t<0 and p2 is returned for values of t > 1.

double eval(const gstPoint& pt) const;
Gives the parametric value t of the line segment for the projection of pt onto the line segment.

virtual const gstPoint& getPointOnLine() const;
Returns a point on the line.

gstPoint origin() const;
Returns p1.

virtual const gstPoint& pointOnLine() const;
Returns a point on the line.

double projectToParametric(const gstPoint& pt) const;
Projects pt onto the line segment and returns the parametric value of the projection.

virtual gstVector unitDirection() const;
GstVector directed along line in the direction of v1 or toward p2 depending on constructor used.

class gstVector

Summary

#include “gstVector.h”
class gstVector : public gstPoint;

Description

3D vector class (x,y,z).

Public constructors

gstVector() ;
Constructor.

gstVector(const gstPoint& point) ;
Constructor.

gstVector(const gstPoint* point) ;
Constructor.

gstVector(const gstVector& v) ;
Constructor.

gstVector(double array [3]) ;
Constructor.

gstVector(double x ,
double y ,
double z) ;
Constructor.

Public Operators

gstVector& operator=(const gstVector& p) ;
Assignment operator.

gstVector& operator=(const gstVector* p) ;
Assignment operator.

gstVector& operator*=(double d) ;
Multiply and assign operator.

gstVector operator+(const gstVector& p) const;
Addition operator.

gstVector& operator+=(const gstVector& p) ;
Accumulation (add and assign) operator.

gstVector operator-() const;
Subtraction operato.r.

gstVector operator-(const gstVector& p) const;
Substraction operator.

gstVector& operator-=(const gstVector& p) ;
Subtract and assign operator.

gstVector& operator/=(double d) ;
Divide and assign operator.

Public Members

gstVector cross(const gstVector& a) ;
Returns cross product of current vector and vector a.

double distance(const gstPoint& a) ;
Returns the magnitude of projection from point a onto vector.

double dot(const gstVector& a) ;
Returns dot product of current vector and vector a.

int getLongestAxisComponent() const;
Return number of longest axis component of vector (0=x, 1=y, 2=z).

int getSecondLongestAxisComponent() const;
Return number of longest axis component of vector (0=x, 1=y, 2=z).

int getShortestAxisComponent() const;
Return number of shortest axis component of vector (0=x, 1=y, 2=z).

double norm() const;
Return vector magnitude.

gstVector& normalize() ;
Normalize vector.

double normalizeReturnNorm() ;
Normalize and return norm.

class gstVertex

Summary

#include “gstVertex.h”
class gstVertex : public gstPoint;

Description

Mesh element representing a position in space with incident gstEdges directed into and out of the vertex.

Public constructors

gstVertex(const gstPoint& _position ,const gstVertexKey _key = NULL) ;

virtual ~gstVertex() ;

Public Members

gstVertexKey getKey() const;
Returns unique key identifying this vertex.

int getNumIncidentEdges() const;
Returns the number of gstEdge elements incident upon this vertex.

gstIncidentEdgeListConstIterator incidentEdgesBegin() const;
Returns iterator positioned at the beginning of a list of incident gstEdges upon this vertex.

gstIncidentEdgeListConstIterator incidentEdgesEnd() const;
Returns iterator positioned at the ending of a list of incident gstEdges upon this vertex.

gstBoolean isStranded() const;
Returns TRUE if no edges are incident upon vertex.

Protected members

void addIncidentEdge(gstEdge* edge) ;
Adds edge to list of incident edges along with direction of edge into or out of vertex.
NOTE: gstEdge::gstEdge should be the only place this method is called from.

gstBoolean removeIncidentEdge(gstEdge* edgeToRemove) ;
Removes edge edgeToRemove from this gstVertex. If edgeToRemove was not incident then FALSE is returned. If edgeToRemove was the only incident edge then gstVertex::STRANDED is returned. Otherwise, TRUE is returned.

Function operator*

Summary #include "gstQuaternions.h"
```
gstQuaternion operator*(const gstQuaternion& q1 ,const gstQuaternion& q2 )
;
```

Description Multiply operator for quats.

Function operator<<

Summary #include "gstPoint.h"

```
ostream& operator<<(ostream& os ,const gstPoint& pt ) ;
```

Function template<gstPoint2D, __default_alloc_template< 1, 0>>operator<<

Summary #include "gstPoint2D.h"
```
ostream& operator<<(ostream& os ,gstPoint2D& pt ) ;
```

Description Declare STL routines used by GHOST to be exported symbols from the dll.

Function operator>>

Summary #include "gstPoint.h"

```
istream& operator>>(istream& os ,gstPoint& pt ) ;
```

Function matrixToQuaternion

Summary #include "gstQuaternions.h"

```
void matrixToQuaternion(double (* r ) [ 3 ] ,gstQuaternion& q ) ;
```

Function multQuaternions

Summary #include "gstQuaternions.h"
`gstQuaternion multQuaternions(const gstQuaternion& q1 ,const gstQuaternion& q2 ) ;`

Description Multiply 2 and return result.

Summary

```
#include "gstQuaternions.h"
void multQuaternions(const gstQuaternion& q1 ,
                     const gstQuaternion& q2 ,
                     gstQuaternion& result ) ;
```

Description Multiply 2 quats, put result in 3rd argument.

Function normalizeQuaternion

Summary #include "gstQuaternions.h"
```
void normalizeQuaternion(gstQuaternion& q1 ) ;
```

Description Normalize a quat.

Function quaternionToMatrix

Summary #include "gstQuaternions.h"
`void quaternionToMatrix(const gstQuaternion& q ,double (* r ) [ 3 ] ) ;`

Description Convert a quat to a 3x3 matrix.

Function scaleQuaternion

Summary #include "gstQuaternions.h"
```
void scaleQuaternion(double s ,gstQuaternion& q ) ;
```

Description Scale quat by a scalar.

Summary

```
#include "gstPoint.h"
gstBoolean withinEpsilon(const gstPoint& pt1 ,
                         const gstPoint& pt2 ,
                         double epsilon  = 0. 00000001) ;
```

Description Check if two points are nearly the same.

Function withinEpsilon

Summary

#include "gstPoint2D.h"

```
gstBoolean withinEpsilon(const gstPoint2D& pt1 ,
                         const gstPoint2D& pt2 ,
                         double epsilon  = 0. 00000001) ;
```

Description Check if 2D points are nearly the same.

class gstBoundary

Summary

#include “gstBoundary.h”
class gstBoundary : public gstShape;

Description

Base boundary class. Keep the PHANTOM inside this shape.

Public constructors

virtual ~gstBoundary() ;
Destructor.

Public Members

virtual gstNode* Clone() const;
Clone.

gstBoundary* CloneBoundary() const;
Clone as gstBoundary.

virtual gstBoolean checkIfPointIsInside_WC(const gstPoint& pt) ;
For internal use.

virtual gstBoolean collisionDetect(gstPHANToM* PHANToM) ;
For extension: Returns TRUE if a line drawn from the previous to the current position of the PHANToM intersects the boundary.

virtual gstType getTypeId() const;
Virtual form of getClassTypeId.

virtual gstBoolean intersection(const gstPoint& startPt_WC ,
const gstPoint& endPt_WC ,
gstPoint& intersectionPt_WC ,
gstVector& intersectionNormal_WC ,
void**) ;
For extension: Checks to see if line segment intersects the boundary. TRUE is returned if the line segment defined by startPt_WC and endPt_WC intersects the boundary. If so, intersectionPt_WC is set to the point of intersection and intersectionNormal_WC is set to surface normal at intersection point.

virtual gstBoolean isOfType(gstType type) const;
Virtual form of staticIsOfType.

virtual void putInSceneGraph() ;
For extension: Used by system or for creating sub-classes only. Puts boundary object in scene graph. Note that gstShapes::putInSceneGraph and removeFromSceneGraph must be skipped so that this object is not put in the shape list. This is a special shape type which is only felt by an identified gstPHANToM object and should not be considered a regular geometry object.

virtual void removeFromSceneGraph() ;
For extension:
Used by system or for creating sub-classes only.
Remove boundary object from scene graph.

static gstType getClassTypeId() ;
Static: get type id of this class.

static gstBoolean staticIsOfType(gstType type) ;
Return TRUE if class is of the given type or is derived from that type.

Protected constructors

gstBoundary() ;
Constructors are protected, use class as parent only Constructor.

gstBoundary(const gstBoundary& origBoundaryNode) ;
Copy Constructor.

gstBoundary(const gstBoundary* origBoundaryNode) ;
Constructor.

class gstBoundaryCube

Summary

#include “gstBoundaryCube.h”
class gstBoundaryCube : public gstBoundary;

Description

Bounding cube boundary class. This restricts the PHANToM inside a box-shaped volume. The box is by default located at the origin with the width along the X-axis, height along the Y-axis, and depth (length) along the Z-axis.

Public constructors

gstBoundaryCube() ;
Constructor.

gstBoundaryCube(const gstBoundaryCube& cube) ;
Copy Constructor.

virtual ~gstBoundaryCube() ;
Destructor.

Public Members

virtual gstNode* Clone() const;
Clone.

gstBoundaryCube* CloneBoundaryCube() const;

virtual gstBoolean checkIfPointIsInside_WC(const gstPoint& pt) ;
For internal use.

virtual gstBoolean collisionDetect(gstPHANToM* PHANToM) ;
For extension: Returns TRUE if a line drawn from the previous to the current position of the PHANToM intersects the boundary.

double getHeight() ;
Get height (Y-axis) of boundary [millimeters].

double getLength() ;
Get depth/length (Z-axis) of boundary [millimeters].

virtual gstType getTypeId() const;
Virtual form of getClassTypeId.

double getWidth() ;
Get width (X-axis) of boundary [millimeters].

virtual gstBoolean intersection(const gstPoint& startPt_WC ,
const gstPoint& endPt_WC ,
gstPoint& intersectionPt_WC ,
gstVector& intersectionNormal_WC ,
void**) ;
For extension: Checks to see if line segment intersects the box boundary. TRUE is returned if the line segment defined by startPt_WC and endPt_WC intersects the boundary. If so, intersectionPt_WC is set to the point of intersection and intersectionNormal_WC is set to surface normal at intersection point.

virtual gstBoolean isOfType(gstType type) const;
Virtual form of staticIsOfType.

void setHeight(double newHeight) ;
Set height (Y-axis) of boundary [millimeters].

void setLength(double newLength) ;
Set depth/length (Z-axis) of boundary [millimeters].

void setWidth(double newWidth) ;
Set width (X-axis) of boundary [millimeters].

static gstType getClassTypeId() ;
Static: get type id of this class.

static gstBoolean staticIsOfType(gstType type) ;
Return TRUE if class is of the given type or is derived from that type.

Protected Data

double length, width, height

class gstCone

Summary

#include “gstCone.h”
class gstCone : public gstShape;

Description

Cone primitive. This class represents the geometry of a cone. The default size orientation of the cone is centered at the origin with the tip pointing up the y-axis with height of 2.0 and radius of 1.0.

Public Constants

NOT_TOUCHED
BASE_TOUCHED
SIDE_TOUCHED

Public constructors

gstCone() ;
Constructor.

gstCone(const gstCone& cone) ;
Copy Constructor.

~gstCone() ;
Destructor.

Public Members

virtual gstNode* Clone() const;
Clone.

gstCone* CloneCone() const;

virtual int checkIfPointIsInside_WC(const gstPoint& pt) ;
For internal use.

virtual int collisionDetect(gstPHANToM* PHANToM) ;
For extension: Used by system or for creating sub-classes only. Returns TRUE if PHANToM is currently in contact with this object. If so, the collision is added to the PHANToM's list through gstPHANToM::getCollisionInfo() and gstPHANToM::collisionAdded().

double getHeight() const;
Get height of object [millimeters].

double getRadius() const;
Get radius of object [millimeters].

virtual gstType getTypeId() const;
Virtual form of getClassTypeId.

virtual gstBoolean intersection(const gstPoint& startPt_WC ,
const gstPoint& endPt_WC ,
gstPoint& intersectionPt_WC ,
gstVector& intersectionNormal_WC ,
void**) ;
For extension: Used by system or for creating sub-classes only. Returns TRUE if line segment defined by startPt_WC and endPt_WC intersects this shape object. If so, intersectionPt_WC is set to point of intersection and intersectionNormal_WC is set to surface normal at intersection point.

virtual gstBoolean isOfType(gstType type) const;
Virtual form of staticIsOfType.

void printSelf(FILE* fp) ;
For internal use.

void setHeight(double newHeight) ;
Set height of object [millimeters].

void setRadius(double newRadius) ;
Set radius of object [millimeters].

static gstType getClassTypeId() ;
Static: get type id of this class.

static gstBoolean staticIsOfType(gstType type) ;
Return TRUE if class is of the given type or is derived from that type.

class gstCube

Summary

#include “gstCube.h”
class gstCube : public gstShape;

Description

Cube primitive. This class represents the geometry of a cube. The default size orientation is centered at the origin with all side lengths of 2.0.

Public constructors

gstCube() ;
Constructor.

gstCube(const gstCube& cube) ;
Constructor.

virtual ~gstCube() ;
Destructor.

Public Members

virtual gstNode* Clone() const;
Clone.

gstCube* CloneCube() const;

virtual int checkIfPointIsInside_WC(const gstPoint& pt) ;
For internal use.

virtual int collisionDetect(gstPHANToM* PHANToM) ;
For extension: Used by system or for creating sub-classes only. Returns TRUE if PHANToM is currently in contact with this object. If so, the collision is added to the PHANToM's list through gstPHANToM::getCollisionInfo() and gstPHANToM::collisionAdded().

double getHeight() const;
Get height of object [millimeters] along y axis.

double getLength() const;
Get length of object [millimeters] along z axis.

virtual gstType getTypeId() const;
Virtual form of getClassTypeId.

double getWidth() const;
Get width of object [millimeters] along x axis.

virtual gstBoolean intersection(const gstPoint& startPt_WC ,
const gstPoint& endPt_WC ,
gstPoint& intersectionPt_WC ,
gstVector& intersectionNormal_WC ,
void**) ;
For extension: Used by system or for creating sub-classes only. Returns TRUE if line segment defined by startPt_WC and endPt_WC intersects this shape object. If so, intersectionPt_WC is set to point of intersection and intersectionNormal_WC is set to surface normal at intersection point.

virtual gstBoolean isOfType(gstType type) const;
Virtual form of staticIsOfType.

void setHeight(double newHeight) ;
Set height of object [millimeters] along y axis.

void setLength(double newLength) ;
Set length of object [millimeters] along z axis.

void setWidth(double newWidth) ;
Set width of object [millimeters] along x axis.

static gstType getClassTypeId() ;
Static: get type id of this class.

static gstBoolean staticIsOfType(gstType type) ;
Return TRUE if class is of the given type or is derived from that type.

Protected Data

gstBoolean	beenOutside
double	height
double	length
gstPoint	old_pos
double	width

class gstCylinder

Summary

#include “gstCylinder.h”
class gstCylinder : public gstShape;

Description

Cylinder primitive. This class represents the geometry of a cylinder. The default position and orientation is centered at the origin with height of 2.0 along y-axis and radius of 1.0.

Public Constants

NOT_TOUCHED
TOP_TOUCHED
SIDE_TOUCHED
BASE_TOUCHED

Public constructors

gstCylinder() ;
Constructor.

gstCylinder(const gstCylinder& cyl) ;
Constructor.

virtual ~gstCylinder() ;
Destructor.

Public Members

virtual gstNode* Clone() const;
Clone.

gstCylinder* CloneCylinder() const;

virtual int checkIfPointIsInside_WC(const gstPoint& pt) ;
For internal use.

virtual int collisionDetect(gstPHANToM* PHANToM) ;
For extension: Used by system or for creating sub-classes only. Returns TRUE if PHANToM is currently in contact with this object. If so, the collision is added to the PHANToM's list through gstPHANToM::getCollisionInfo() and gstPHANToM::collisionAdded().

double getHeight() const;
Get height of object [millimeters].

double getRadius() const;
Get radius of object [millimeters].

virtual gstType getTypeId() const;
Virtual form of getClassTypeId.

virtual gstBoolean intersection(const gstPoint& startPt_WC ,
const gstPoint& endPt_WC ,
gstPoint& intersectionPt_WC ,
gstVector& intersectionNormal_WC ,
void**) ;
For extension: Used by system or for creating sub-classes only. Returns TRUE if line segment defined by startPt_WC and endPt_WC intersects this shape object. If so, intersectionPt_WC is set to point of intersection

and intersectionNormal_WC is set to surface normal at intersection point.

virtual gstBoolean isOfType(gstType type) const;
Virtual form of staticIsOfType.

void printSelf(FILE* fp) ;
For internal use.

void setHeight(double newHeight) ;
Set height of object [millimeters].

void setRadius(double newRadius) ;
Set radius of object [millimeters].

static gstType getClassTypeId() ;
Static: get type id of this class.

static gstBoolean staticIsOfType(gstType type) ;
Return TRUE if class is of the given type or is derived from that type.

class gstShape

Summary #include “gstShape.h”