GHOST® SDK

API Reference

Version 4.0

SensAble Technologies, Inc. ®

.Copyright Notice

Phone: (888) SENSABLE
e-mail support@sensable.com
http://www.sensable.com

Trademarks

- Windows NT and Visual C++ are registered trademarks of Microsoft Corporation.
- GHOST and PHANTOM are registered trademarks of SensAble Technologies, Inc.
- 3D Touch and SensAble are trademarks of SensAble Technologies, Inc.
- IRIX, OpenGL, Open Inventor, and Indigo2 are trademarks of Silicon Graphics Incorporated.
- Netscape is a registered trademark of Netscape, Inc.

Disclaimer

SensAble Technologies does not warrant that this product or its accompanying documentation is error free; they are subject to change without notice.

Updates

Please check the SensAble (www.sensable.com) web site for updated information on the GHOST SDK.

GHOST SDK Reference Manual

April 29, 2002

GHOST SDK Reference Manual

Section 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) ;

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 ) ;
```

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.

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 ) ;
```

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”
class gstShape : public gstBoundedHapticObj;

Description

Base class for haptic geometry nodes.

Public constructors

virtual ~gstShape() ;
Destructor.

Public Members

virtual gstNode* Clone() const;
Clone.

gstShape* CloneShape() const;

gstBoolean addCollision(gstPHANToM* PHANToM ,
gstPoint& SCP ,
gstVector& SCPnormal) ;
For extension: Used by system or for creating sub-classes only. Takes the current surface contact point (SCP) and surface contact point normal (SCPnormal), transforms them to the world coordinate system, and adds them to gstPHANToM's list of current collisions.

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

virtual gstBoolean collisionDetect(gstPHANToM* phantomNode) ;
For extension: Used by system or for creating sub-classes only. Returns TRUE if the 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().

virtual void cornerCheck() ;
For internal use.

virtual gstPoint fromParent(const gstPoint& p) ;
Transforms 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) ;
Transforms vector v, which is in the parent coordinate reference frame, to the vector in the local coordinate reference frame.

virtual gstPoint fromParentNoLocalScale(const gstPoint& p) ;
For extension:
Transforms point p, which is in the parent coordinate reference frame, to the point in the local coordinate reference frame - scale for this node.

virtual gstVector fromParentNoLocalScale(const gstVector& v) ;
For extension:

Transforms vector v, which is in the parent coordinate reference frame, to the vector in the local coordinate reference frame - scale for this node.

virtual gstPoint fromWorld(const gstPoint& p) ;
Transforms 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) ;
Transforms vector v, which is in the world coordinate reference frame, to the vector in the local coordinate reference frame.

virtual gstPoint fromWorldNoLocalScale(const gstPoint& p) ;
For extension:
Transforms point p, which is in the world coordinate reference frame, to the point in the local coordinate reference frame - scale for this node.

virtual gstVector fromWorldNoLocalScale(const gstVector& v) ;
For extension:
Transforms vector v, which is in the world coordinate reference frame, to the vector in the local coordinate reference frame - scale for this node.

virtual gstTransformMatrix& getCumulativeTransform() ;
Used by system or for creating sub-classes only. Returns cumulative transformation matrix.

gstShape* getNextNearShapeInScene() ;
For internal use.

gstShape* getNextShapeInScene() ;
Returns "next" shape in shapes scene list.

virtual int getStateForPHANToM(gstPHANToM* curPHANToM) ;
For extension: Return integer representing the contact state for curPHANToM with this shape node. 0 is assumed to be FALSE and represents no contact between curPHANToM and this shape node. Otherwise, the value may indicate special information about the contact between curPHANToM this shape node.

double getSurfaceFdynamic() const;
Get surface dynamic friction coefficient.

double getSurfaceFstatic() const;
Get surface static friction coefficient.

double getSurfaceKdamping() const;
Get surface damping coefficient.

double getSurfaceKspring() const;
Get surface spring constant.

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** data) ;
For extension: Used by system or for creating sub-classes only. Returns TRUE if the line segment, defined by startPt_WC and endPt_WC in the world coordinate system, intersects the shape object. If TRUE, intersectionPt_WC is set to the point of intersection and intersectionNormal_WC is set to the surface normal at the intersection point.

virtual void invalidateCumTouchability() ;
The node needs to reset its contacts when its touchability state changes so that PHANToMs can no longer reference it.

gstBoolean isInContact() ;
Returns TRUE if object is in contact with any PHANToM in the scene.

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 the scene graph.

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

void resetPHANToMContacts() ;
Resets all contact state pertaining to gstPHANToMs. Thus, any gstPHANToMs in contact with this shape will no longer be considered in contact.

virtual void setCenter(const gstPoint&) ;
For internal use.

virtual void setCornerData(void* data) ;
For internal use.

virtual void setEdgeData(void* data) ;
For internal use.

void setSurfaceFdynamic(double newFd) ;
Set surface dynamic friction coefficient. Range (0..1.0).

void setSurfaceFstatic(double newFs) ;
Set surface static friction coefficient.
Range (0..1.0)

void setSurfaceKdamping(double newKd) ;
Set surface damping coefficient [Kg/(1000.0*sec)]. Range (0..0.005).

void setSurfaceKspring(double newKs) ;
Set surface spring constant [Kg/(1000.0*sec^2)]. Range (0..1.0).

virtual void setTouchableByPHANToM(gstBoolean bTouchable) ;
If flag is TRUE, shape is able to have contact with any gstPHANToMs in the scene graph. Otherwise, shape becomes transparent to gstPHANToMs in the scene graph.

virtual gstPoint toParent(const gstPoint& p) ;
Transforms 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) ;
Transforms vector v, which is in the local coordinate reference frame, to the vector in the parent coordinate reference frame.

virtual gstPoint toParentNoLocalScale(const gstPoint& p) ;
For extension: Transforms point p, which is in the local coordinate reference frame - scale for this node, to the point in the parent coordinate reference frame.

virtual gstVector toParentNoLocalScale(const gstVector& v) ;
For extension: Transforms vector v, which is in the local coordinate reference frame - scale for this node, to the vector in the parent coordinate reference frame.

virtual gstPoint toWorld(const gstPoint& p) ;
Transforms 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) ;
Transforms vector v, which is in the local coordinate reference frame, to the vector in the world coordinate reference frame.

virtual gstPoint toWorldNoLocalScale(const gstPoint& p) ;
For extension:
Transforms point p, which is in the local coordinate reference frame - scale for this node, to the point in the world coordinate reference frame.

virtual gstVector toWorldNoLocalScale(const gstVector& v) ;
For extension:
Transforms vector v, which is in the local coordinate reference frame - scale for this node, to the vector in the world coordinate reference frame.

virtual gstBoolean updateStateForPHANToM(gstPHANToM* curPHANToM ,int inContact) ;
For extension: Update the contact state between this shape node and curPHANToM. This stores the state for later retrieval with getStateForPHANToM.

static void addNearShapeInScene(gstShape* nextNear) ;
For internal use.

static void clearNearShapesInScene() ;
For internal use.

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

static double getDefaultSurfaceFdynamic() ;
Get default surface dynamic friction coefficient.

static double getDefaultSurfaceFstatic() ;
Get default surface static friction coefficient.

static double getDefaultSurfaceKdamping() ;
Get default surface damping coefficient.

static double getDefaultSurfaceKspring() ;
Get default surface spring constant [Kg/(1000.0*sec^2)].

static gstShape* getNearShapesInSceneHead() ;
For internal use.

static gstShape* getShapesInScene() ;
Returns first shape in shapes scene list.

static void setDefaultSurfaceFdynamic(double newFd) ;
Set default surface dynamic friction coefficient. Range (0..1.0).

static void setDefaultSurfaceFstatic(double newFs) ;
Set default surface static friction coefficient. Range (0..1.0).

static void setDefaultSurfaceKdamping(double newKd) ;
Set default surface damping coefficient [Kg/(1000.0*sec)]. Range (0..0.005).

static void setDefaultSurfaceKspring(double newKs) ;
Set default surface spring constant [Kg/(1000.0*sec^2)]. Range (0..1.0).

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

Public Data	static const gstEventType	TOUCHED
	static const gstEventType	UNTOUCHED
Protected constructors		gstShape() ; This class is intended as a base class only, the constructors are protected so that instances can not be created.
		gstShape(const gstShape& origShapeNode) ;
		gstShape(const gstShape* origShapeNode) ;
Protected Data	gstBoolean	_resetPHANToMContacts
	static double	defaultSurfaceFd
	static double	defaultSurfaceFs
	static double	defaultSurfaceKd
	static double	defaultSurfaceKs - These are the static default values used to set above params in constructor.
	static gstShape*	nearShapesInSceneHead
	gstShape*	nextNearShapeInScene
	gstShape*	nextShapeInScene
	static gstShape*	shapesInSceneHead
	gstShapeStateArrayStruct*	stateArray - For extension: Contact state array.
	double	surfaceFd - For internal use.
	double	surfaceFs - For internal use.
	double	surfaceKd - For internal use.
	double	surfaceKs - For internal use.

class gstSphere

Summary

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

Description

Sphere primitive. This class represents the geometry of a sphere.

Public constructors

gstSphere() ;
Constructor.

gstSphere(const gstSphere& sphere) ;
Constructor.

virtual ~gstSphere() ;
Destructor.

Public Members

virtual gstNode* Clone() const;
Clone.

gstSphere* CloneSphere() 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 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 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 gstTorus

Summary

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

Description

Torus primitive. This represents the geometry of a torus. The default position and orientation is centered about the origin with the ring radius (Major Radius) set to 2/3 and the swept circle radius (Minor Radius) set to 1/3. The y-axis passes through the hole of the Torus.

Public constructors

gstTorus() ;
Constructor.

gstTorus(const gstTorus& torus) ;
Constructor.

virtual ~gstTorus() ;
Destructor.

Public Members

virtual gstNode* Clone() const;
Clone.

gstTorus* CloneTorus() 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 getMajorRadius() const;
Get major radius [millimeters].

double getMinorRadius() const;
Get minor radius [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:
 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 setMajorRadius(double newRadius) ;
Set major radius [millimeters].

void setMinorRadius(double newRadius) ;
Set minor radius [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.

Chapter 3. Dynamics

class gstButton

Summary

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

Description

Dynamic button class. This creates a button which is aligned down the z-axis, with the nominal position (resting) at the origin. The button can be pushed from the origin along the negative z-axis until the end of travel (-throwDist). For a small region (deadband) before the end of travel, the button spring becomes inactive and only the much smaller restoring force is used. This produces a feeling of a soft click when the button is pushed into the deadband region. If the user releases the button, it should move back to the origin. GstButton produces a pressed or released event when the button transitions between these two states. The id field of the gstEvent structure passed into the event callback is assigned gstButton::PRESSED or gstButton::RELEASED for the corresponding event.

Public constructors

gstButton() ;
Constructor.

gstButton(const gstButton& button) ;
Copy Constructor.

virtual ~gstButton() ;
Destructor.

Public Members

virtual gstNode* Clone() const;
Clone.

gstButton* CloneButton() const;

double getDeadband() const;
Get deadband [millimeters].

gstEvent getEvent() ;
Get current state of object.

double getK() const;
Get spring constant [Kg/(1000.0*sec^2)].

double getRestoringForce() const;
Get restoring force [Newtons].

double getThrowDist() const;
Get throw distance [millimeters].

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

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

void setDeadband(double _deadband) ;
Set deadband [millimeters].

void setK(double _K) ;
Set spring constant [Kg/(1000.0*sec^2)].

void setRestoringForce(double _restoringForce) ;
Set restoring force [Newtons] to push button out of the deadband region. This force is only active when the button is in the deadband region.

void setThrowDist(double _throwDist) ;
Set throw distance [millimeters].

virtual void updateDynamics() ;
```
For extension:
Used by system or for creating sub-classes only.
When button has velocity or a force, this method
does Euler integration of button state and
creates button events to be processed by
gstTransform::updateEvents().
```

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.

Public Data

static const gstEventType	PRESSED
static const gstEventType	RELEASED

Protected Data

double	K - Spring constant for button.
gstEventType	currentState
double	deadband - Length of deadband at end of button throw.
double	restoringForce - Small restoring force used to push button through deadband.
double	throwDist - Distance button moves.

class gstDial

Summary

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

Description

Dial dynamic class. The dial rotates in the z=0 plane and is by default located at the origin. The dial has notches (detents), a moment of inertia, a damping coefficient, and a spring constant associated with the amount of force needed to leave a notch. Notches are placed evenly spaced from 0-359 degrees with notch 0 starting at 0 degrees. Methods to set the number of notches, the dial's location relative to these notches, and the spring constant are available in this class. Other state variables (e.g., damping coefficient) are inherited from the base class, gstDynamic. GstDial produces a notch event each time the dial moves into a new notch. The id field of the gstEvent structure passed the event callback is assigned the current notch and the data1d integer field specifies the direction the dial came from by assigning the field GST_COUNTERCLOCKWISE or GST_CLOCKWISE.

Public constructors

gstDial() ;
Constructor.

gstDial(const gstDial& dial) ;
Copy Constructor.

virtual ~gstDial() ;
Destructor.

Public Members

virtual gstNode* Clone() const;
Clone.

gstDial* CloneDial() const;

gstEvent getEvent() ;
Get current state of object.

int getInitialNotch() const;
Get initial notch (angular orientation) of dial.

double getK() const;
Get spring constant of spring that is holding dial in current notch [Kg/(1000.0*sec^2)].

int getNumberNotches() const;
Get number of notches (detents).

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

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

void setInitialNotch(int initNotch) ;
Set initial notch (angular orientation) of dial.

void setK(double _K) ;
Set spring constant of spring that is holding dial in current notch [Kg/(1000.0*sec^2)].

void setNumberNotches(int n) ;
Set the number of notches (detents). Notches are placed evenly spaced from 0-359 degrees with notch 0 starting at 0 degrees.

virtual void updateDynamics() ;
```
For extension:
Used by system or for creating sub-classes only.
When button has velocity or a force, this method
does Euler integration of button state and
creates button events to be processed by
gstTransform::updateEvents().
```

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.

Public Data

static const gstEventType	CLOCKWISE
static const gstEventType	COUNTERCLOCKWISE

Protected Data

int number_notches,	currentNotch
gstEventType	direction
int	initial_notch
double	notch
double K,	orientation
double	sectionTheta - Angle of each sectionTheta.

class gstDynamic

Summary

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

Description

Base class for dynamic nodes.

Public constructors

virtual ~gstDynamic() ;
Destructor.

Public Members

virtual void addChild(gstTransform* newChild) ;
Make newChild a child of this node.

void addForceTorque(const gstVector& force_WC ,gstPoint& SCP_WC) ;
For extension: Used by system or for creating sub-classes only. Add force and torque pair if force is above threshold and node is in scene. Force and torque calculated from this call are used in the next call to updateDynamics.

void addToDynamicList() ;
For extension: Used by system or for creating sub-classes only. Add object to list of gstDynamic objects needing dynamic update. GstDynamic nodes in this list have their updateDynamics method called every servoLoop until removed by removeFromDynamicList. Objects are usually only put on this list if they have a sufficient force imposed on them and stay on until their velocity falls below a threshold value for a certain amount of time (1 second).

gstBoolean dynamicMoveThisServoLoop() const;
For internal use.

gstPoint fromWorldLast(const gstPoint& p) ;
For internal use.

gstVector fromWorldLast(const gstVector& v) ;
For internal use.

const gstVector& getAccel() const;
Get acceleration [millimeters/(second squared)].

const gstVector& getAngularAccel() const;
Get angular acceleration in [radians/(second squared)].

const gstVector& getAngularVelocity() const;
Get angular velocity [radians/second].

double getDamping() const;
Get damping coefficient [Kg/(1000.0*sec)].

gstTransformMatrix& getLastCumulativeTransform() ;
For internal use.

double getMass() const;
Get mass [kilograms].

const gstVector& getReactionForce() const;
Get reaction force from PHANToM [Newtons].

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

const gstVector& getReactionTorque() const;
Get reaction torque from PHANToM [Newton*millimeters].

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

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

const gstVector& getVelocity() const;
Get velocity [millimeters/second].

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 put in scene graph.

virtual void removeChild(gstTransform* childToRemove) ;
Remove child.

virtual gstTransform* removeChild(int childIndex) ;
Remove child.

void removeFromDynamicList() ;
For extension:
Used by system or for creating sub-classes only.
Remove object from list of dynamic objects needing dynamic update.

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

graph.

virtual void resetDynamicState() ;
For extension:
Reset dynamic state of gstDynamic. Subclasses should overload to reset their own dynamic state and call gstDynamic::resetState before returning.

void rotateDynamic(const gstVector& axis ,double rad) ;
For extension: Accumulate rotation with previous rotation of dynamic object.

void scaleDynamic(double scale) ;
For extension: Accumulate scale with previous scale of dynamic object.

void setAccel(const gstVector newAccel) ;
Set acceleration [millimeters/(second squared)].

void setAngularAccel(const gstVector newAccel) ;
Set angular acceleration [radians/(second squared)].

void setAngularVelocity(const gstVector newVel) ;
Set angular velocity [radians/second].

void setDamping(double newDamping) ;
Set damping coefficient [Kg/(1000.0*sec)].

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

void setMass(double newMass) ;
Set mass [kilograms].

void setPositionDynamic(const gstPoint& newPos) ;
For extension: Overwrite previous position with new position for dynamic object.

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

void setRotationDynamic(const gstVector& axis ,double rad) ;
For extension: Overwrite previous rotation with new rotation for dynamic object.

void setScaleDynamic(double newScale) ;
For extension: Overwrite previous scale with new scale for dynamic object.

void setTransformMatrixDynamic(const gstTransformMatrix& matrix) ;
For extension: Set homogenous transformation matrix for dynamic object.

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

void setTranslationDynamic(const gstPoint& translation) ;
For extension: Overwrite previous translation with new translation for dynamic object.

void setVelocity(const gstVector newVel) ;
Set velocity [millimeters/second].

void setVelocity(double x ,
double y ,
double z) ;
Set velocity [millimeters/second].

gstPoint toWorldLast(const gstPoint& p) ;
For internal use.

gstVector toWorldLast(const gstVector& v) ;
For internal use.

void translateDynamic(const gstPoint& translation) ;
For extension: Accumulate new translation with previous translation of dynamic object.

virtual void updateDynamics() ;
```
For extension:
Used by system or for creating sub-classes only.
When subclassing, call gstDynamic::updateDynamics() at the
end of your updateDynamics() procedure.
```

void updateLastObjTransf() ;
For internal use.

void useCalculatedAccel() ;
For internal use.

void useConstantAccel(const gstVector& accel) ;
For internal use.

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

static double getDefaultDamping() ;
Get default damping coefficient [Kg/(1000.0*sec)].

static double getDefaultMass() ;
Get default mass [kilograms].

static double getDeltaT() ;
Get length of time between calls to updateDynamics() [seconds].

static void nextServoLoop() ;
```
For extension:
Used by system or for creating sub-classes only.
Do cleanup to prepare for next servoLoop
```

static void setDefaultDamping(double newDamping) ;
Set default damping coefficient [Kg/(1000.0*sec)].

static void setDefaultMass(double newMass) ;
Set default mass [kilograms].

static void setDeltaT(double newDeltaT) ;
For internal use.

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

static void staticUpdateDynamics() ;
For internal use.

Protected constructors

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

gstDynamic(const gstDynamic& origDynamicNode) ;
Copy Constructor.

gstDynamic(const gstDynamic* origDynamicNode) ;

Protected Data

gstVector	acceleration - For extension: mm/s^2.
gstVector	angularAccel - For extension: rad/s^2.
gstVector	angularVel - For extension: rad/s.
double	damping - For extension: damping constant.
static double	defaultDamping - These are the default values used to set above params in constructor.
static double	defaultMass
static double	deltaT - For extension: 1/updateRate.
static gstDynamic*	dynamicListHead
int	inDynamicList
double	mass - For extension: mass of dynamic object represented by sub-tree below this node.
gstVector	reactionForce - For extension: Newtons.
gstVector	reactionTorque - For extension: milli-Newtons/m.
int	removeFromDynamicListCounter
gstDynamic* leftDynamic, *	rightDynamic
gstBoolean	usingConstantAccel
gstVector	velocity - For extension: mm/s.

class gstPHANToMDynamic

Summary

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

Description

PHANToM haptic interface dynamic class. This node has a PHANToM device associated with it, much like the gstPHANToM node and therefore shares many methods in common with gstPHANToM. Note that the PHANToM encoders are reset by default when a gstPHANToMDynamic node is instantiated. For more information on reseting the PHANToM refer to the gstPHANToM section of the GHOST Programming Guide. The position and orientation of this node are influenced or may correlate directly with the position and orientation of the associated PHANToM device specified by the configuration file passed to the constructor. The descendent geometry nodes of this node have force interactions with other gstPHANToM nodes in scene. For example, a gstCube under a gstTranslateManipulator is translated around to match the translation of the PHANToM device. As gstPHANToM nodes touch the gstCube, forces are sent to the gstPHANToM and the PHANToM associated with this node as if you are moving the block with one PHANToM and preventing it's movement with the gstPHANToM node.

Public constructors

gstPHANToMDynamic(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 ~gstPHANToMDynamic() ;
Destructor.

Public Members

virtual int forcesOff() ;
For internal use.

virtual int forcesOn() ;
For internal use.

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

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

gstEffect* getEffect() ;
Returns current effect currently associated with the PHANToM.

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

void getInfo(gstPHANToMInfoStruct* info) ;
For internal use.

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

double getMaxGain() const;

Returns the max gain value before buzzing will occur.

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].

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

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

gstBoolean getValidConstruction() const;
Returns TRUE is instance had no errors during construction.

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: Called 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 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 simulating. Otherwise, forces will not be turned on nor used at any time.

virtual int setForce_WC(const gstVector& force_WC ,const gstVector& torque_WC) ;
For internal use.

void setMaxGain(double newMaxGain) ;
Returns the max gain value before buzzing will occur.

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

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

int update() ;
For extension:
Used by system or for creating sub-classes only.
Update the PHANToM.

virtual void updateDynamics() ;
For extension: Called every servo loop to update PHANToM state and dynamic state of node.

static int disableAllForces() ;
For internal use.

static int enableAllForces() ;
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 int updateAll() ;
For internal use.

Protected members

void prepareForNextLoop() ;
For internal use.

Protected Data

gstVector PHANToMForce
gstVector PHANToMTorque
gstBoolean _useForces

gstEffect*	effect
gstBoolean	forcesAreOn
static gstPHANToMDynamic*	gstPHANToMDynamicHead
gstPHANToMInfoStruct	info
gstPoint	lastPos
gstPoint	lastPosition_WC
double	maxGain
gstPHANToMDynamic*	nextPHANToMDynamic
int	phantomId
gstTransformMatrix	phantomMatrix - For extension: 4x4 homogenous matrix for PHANToM position and orientation. This is updated automatically, every servo loop.
gstVector	phantomVelocity - For extension: Velocity of PHANToM in mm/sec.
gstBoolean	stylusSwitch
gstBoolean	validConstruction

class gstPHANToMRotation

Summary

#include “gstPHANToMRotation.h”
class gstPHANToMRotation : public gstPHANToMTranslation;

Description

Rigid body PHANToM dynamic class. This sets the PHANToM to control the geometry of its subgraph through rotation. Dynamic forces are sent back to PHANToM simulating a rotating mass with realistic inertial properties. Note that Mass and damping are set through the appropriate methods inherited from gstDynamic.

Public constructors

gstPHANToMRotation(char* initFile) ;
Constructor.

virtual ~gstPHANToMRotation() ;
Destructor.

Public Members

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

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

virtual void updateDynamics() ;

```
For extension:
Used by system or for creating sub-classes only.
When button has velocity or a force, this method
does Euler integration of button state and
creates button events to be processed by
gstTransform::updateEvents().
```

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 gstPHANToMTranslation

Summary

#include “gstPHANToMTranslation.h”
class gstPHANToMTranslation : public gstPHANToMDynamic;

Description

Translation-only PHANToM dynamic class. This sets the PHANToM to control the geometry of its subgraph through translation only. Dynamic forces are sent back to PHANToM to add force feedback information to translation. Note that mass and damping can be modified through methods inherited from gstDynamic.

Public constructors

gstPHANToMTranslation(char* initFile) ;
Constructor.

virtual ~gstPHANToMTranslation() ;
Destructor.

Public Members

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

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

virtual void updateDynamics() ;

```
For extension:
Used by system or for creating sub-classes only.
When button has velocity or a force, this method
does Euler integration of button state and
creates button events to be processed by
gstTransform::updateEvents().
```

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 gstRigidBody

Summary

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

Description

Rigid body dynamic. Simulates an inertial body containing both rotational and linear inertia. Mass and damping are inherited from gstDynamic.

Public constructors

gstRigidBody() ;
Constructor.

~gstRigidBody() ;
Destructor.

Public Members

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

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

void setGravity(gstVector& newGravity) ;
Set gravity (acceleration) [mm/sec^2]. Gravity is in world coordinates Default is 0.0,0.0,0.0.

virtual void updateDynamics() ;
For extension: Used by system or for creating sub-classes only. When body has velocity or a force, this method does Euler integration of rigid body state state.

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	Ibody [3] [3] - Inertia Matrix.
gstVector	L - Angular Momentum.
gstVector	gravity - Gravity.
gstQuaternion q, qDot,	qL

class gstSlider

Summary

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

Description

Slider dynamic class. The slider translates in the z=0 plane and is, initially, located at the origin oriented along the X axis. The slider has notches (or detents), a moment of intertia, a damping coefficient, a throw distance, and a spring constant representing the force needed to leave a notch. Notches are placed evenly along the slider's extent (or throw distance). Methods to set the number of notches, the sliders's location, throw distance, and the spring constant are available. Other state variables (e.g., damping coefficient) are inherited from the base class, gstDynamic.

Public constructors

gstSlider() ;
Constructor.

gstSlider(const gstSlider& slider) ;
Copy Constructor.

virtual ~gstSlider() ;
Destructor.

Public Members

virtual gstNode* Clone() const;
Clone.

gstSlider* CloneSlider() const;

int getClosestNotch() ;
Returns integer ID of nearest notch.

double getDistance() const;
Get slider throw (travel) distance.

gstEvent getEvent() ;
Get current state of object.

int getInitialNotch() const;
Get initial notch.

double getK() const;
Get notch spring constant.

int getNotch() ;
Get current notch.

int getNumberNotches() const;
Get number of notches in slider.

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

virtual gstBoolean isOfType(gstType type) const;

Virtual form of staticIsOfType.

void setDistance(double dist) ;
Set throw (travel) distance of slider [millimeters].

void setInitialNotch(int newNotch) ;
Set initial notch location for slider.

void setK(double _K) ;
Set notch spring constant [Kg/(1000.0*sec^2)].

void setNumberNotches(int n) ;
Set the number of notches (minimum is two--one at each end of slider).

virtual void updateDynamics() ;
For extension: Used by system or for creating sub-classes only. Performs Euler integration and updates state of slider.

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.

Public Data

static const gstEventType	DOWN
static const gstEventType	UP

Protected Data

double	K
double	distance - Total distance traversable by slider.
int	initial_notch
int	notch
int	number_notches
double	sectionLength

Chapter 4. Effects

class gstBuzzEffect

Summary

#include “gstBuzzEffect.h”
class gstBuzzEffect : public gstEffect;

Description

Buzz effect for PHANToM. This effect vibrates the PHANToM end point along the y-axis with a given frequency, amplitude, and duration.

Public constructors

gstBuzzEffect() ;
Constructor.

~gstBuzzEffect() ;
Destructor.

Public Members

virtual gstVector calcEffectForce(void* phantom ,gstVector& torques) ;

virtual gstVector calcEffectForce(void* phantom) ;
For extension: Caculate the force. Force is returned in parent reference frame of phantom. When subclassing, the first parameter should be cast to gstPHANToM to retrieve any information about the state of the PHANToM that is needed to calculate the forces. DeltaT should be used to update the time. Also, if the effect is not active, the zero vector should be returned. ACTUNG! WARNING!: Never call PHANToM->setForce or PHANToM->setForce_WC from this function. It will cause an infinite recursion.

double getAmplitude() ;
Get amplitude of effect [millimeters].

double getDuration() ;
Get duration of effect [seconds].

double getFrequency() ;
Get "buzz" frequency [Hz].

void setAmplitude(double newAmp) ;
Set amplitude of effect [millimeters].

void setDuration(double durInSec) ;
Set duration of effect [seconds].

void setFrequency(double newFreq) ;
Set "buzz" frequency [Hz].

Protected Data

double frequency, amplitude, duration

class gstConstraintEffect

Summary

#include “gstConstraintEffect.h”
class gstConstraintEffect : public gstEffect;

Description

Constraint effect for PHANToM. This effect constrains the PHANToM to a point, line or plane using a spring/damper system. When this effect is started, its force effects ramp up over three seconds so that no significant forces are immediately generated.

Public constructors

gstConstraintEffect() ;
Constructor.

~gstConstraintEffect() ;
Destructor.

Public Members

virtual gstVector calcEffectForce(void* phantom ,gstVector& torques) ;

virtual gstVector calcEffectForce(void* phantom) ;
For extension: Caculate the force. Force is returned in parent reference frame of phantom. When subclassing, the first parameter should be cast to gstPHANToM to retrieve any information about the state of the PHANToM that is needed to calculate the forces. DeltaT should be used to update the time. Also, if the effect is not active, the zero vector should be returned. ACTUNG! WARNING!: Never call PHANToM->setForce or PHANToM->setForce_WC from this function. It will cause an infinite recursion.

double getDamping() const;
Get damping coefficient.

double getSpringStiffness() const;
Get spring constant (stiffness) [Kg/(1000.0*sec^2)].

void setAttenuation(gstBoolean flag) ;

void setDamping(double newDamping) ;
Set damping coefficient.

void setLine(gstPoint newPointOnLine ,gstVector newLineDir) ;
Set a line constraint given a point on the line and a vector.

void setPlane(gstPlane newPlane) ;
Set a planar constraint.

void setPoint(gstPoint newPoint) ;
Set a point constraint.

void setSpringStiffness(double newStiffness) ;
Set spring constant (stiffness) [Kg/(1000.0*sec^2)].

Protected Data

double attenuation
int constraint
double springStiffness, damping

gstVector	lineDirection
gstPlane	planeConstraint
gstPoint	pointConstraint
gstPoint	pointOnLine
gstBoolean	useAttenuation

class gstEffect

Summary

#include “gstEffect.h”
class gstEffect ;

Description

Base class for PHANToM special effects. Unlike other GHOST force phenomena, these effects are geometry-independent, meaning that forces are not generated based on any specific geometric object.

Public constructors

virtual ~gstEffect() ;
Destructor.

Public Members

virtual gstVector calcEffectForce(void* PHANToM ,gstVector& torques) ;

virtual gstVector calcEffectForce(void* PHANToM) ;
This method will protect current customer's code by allowing it to compile and run in the new scheme where the method is called with two arguments.

gstBoolean isActive() const;
Returns TRUE if manip is currently active.

virtual gstBoolean start() ;

```
For extension:
Start the effect.  WARNING:  When re-starting an effect,
make sure to reset any state, such as past PHANToM position.
Otherwise, the next call to calcEffectForce could
generate unexpectedly large forces.
```

virtual void stop() ;

```
For extension:
Stop the effect.
```

Protected constructors

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

Protected Data

gstBoolean active - For extension: TRUE if effect is active.
double time - For extension: Time since start in seconds.

class gstInertiaEffect

Summary

#include “gstInertiaEffect.h”
class gstInertiaEffect : public gstEffect;

Description

Inertia effect for PHANToM. This effect simulates inertia at the endpoint of a PHANToM as if a mass were attached there, using a spring/damper model. The mass, spring constant, and damping coefficient can be specified.

Public constructors

gstInertiaEffect() ;
Constructor.

~gstInertiaEffect() ;
Destructor.

Public Members

virtual gstVector calcEffectForce(void* data ,gstVector& torques) ;

virtual gstVector calcEffectForce(void* data) ;
For extension: Caculate the force. Force is returned in parent reference frame of phantom. When subclassing, the first parameter should be cast to gstPHANToM to retrieve any information about the state of the PHANToM that is needed to calculate the forces. DeltaT should be used to update the time. Also, if the effect is not active, the zero vector should be returned. ACTUNG! WARNING!: Never call PHANToM->setForce or PHANToM->setForce_WC from this function. It will cause an infinite recursion.

double getDamping() const;
Get damping coefficient [Kg/(1000.0*sec)].

gstVector getGravity() ;
Get Gravity in Meters/Sec^2.

double getMass() const;
Get mass [kilograms].

double getSpringStiffness() const;
Get spring constant (stiffness) [Kg/(1000.0*sec^2)].

void setDamping(double newDamping) ;
Set damping coefficient [Kg/(1000.0*sec)].

void setGravity(gstVector _gravity) ;
Set Gravity in Meters/Sec^2.

void setMass(double newMass) ;
Set mass [kilograms].

void setSpringStiffness(double newStiffness) ;
Set spring constant (stiffness) [Kg/(1000.0*sec^2)].

Protected Data

gstVector velocity, acceleration, gravity
gstPoint position

double mass, damping, springStiffness

Chapter 5. Force Fields

class gstConstantForceField

Summary

#include “gstConstantForceField.h”
class gstConstantForceField : public gstForceField;

Description

A particular type of force field. When the user enters the bounding volume associated with this force field, they will feel a force vector as set by setConstantForceVector.

Public constructors

gstConstantForceField() ;
Constructor.

gstConstantForceField(const gstConstantForceField& origForceField) ;
Constructor.

gstConstantForceField(const gstConstantForceField* origForceField) ;
Constructor.

gstConstantForceField(gstVector& force) ;
Constructor.

gstConstantForceField(gstVector& force ,gstVector& torque) ;
Constructor.

virtual ~gstConstantForceField() ;
Destructor.

Public Members

virtual gstNode* Clone() const;
Clone.

gstConstantForceField* CloneForceField() const;

virtual gstVector calculateForceFieldForce(gstPHANToM* phantom) ;
GstForceField method overloaded to define the force applied by the PHANToM when the PHANToM has entered the bounding volume associated with the force field.

virtual gstVector calculateForceFieldForce(gstPHANToM* phantom ,gstVector& torque) ;
GstForceField method overloaded to define the forces and torques applied by the PHANToM when the PHANToM has entered the bounding volume associated with the force field.

gstVector getConstantForceVector() const;
Method to get the force vector that the user will feel whenever the PHANToM enters the bounding volume associated with this force field.

gstVector getConstantTorqueVector() const;
Method to get the torque vector that the user will feel whenever the PHANToM enters the bounding volume associated with this force field.

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

virtual gstBoolean isOfType(gstType type) const;

Virtual form of staticIsOfType.

void setConstantForceVector(const gstVector& force) ;
Method to set the force vector that the user will feel whenever the PHANToM enters the bounding volume associated with this force field.

void setConstantTorqueVector(const gstVector& torque) ;
Method to set the torque vector that the user will feel whenever the PHANToM enters the bounding volume associated with this force field.

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 gstForceField

Summary

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

Description

The GHOST Force Field Class. The class is implemented to allow the developer to prescribe forces to the PHANToM when the user is within the associated bounding volume. The implementation is based on the developer subclassing this class and overloading the calculateForceFieldForce. The forces are integrated with forces generated by the shape objects in the scene.

Public Constants

TOUCHED
UNTOUCHED

Public constructors

virtual ~gstForceField() ;
Destructor.

Public Members

virtual gstNode* Clone() const;
Clone.

gstForceField* CloneForceField() const;

gstVector attenuatedForce(gstPHANToM* phantom ,
gstPoint& PHANToMpos ,
gstVector& appliedTorque) ;
Internal method used to attenuate the force as the user moves away from the force field boundary.

virtual gstVector calculateForceFieldForce(gstPHANToM* phantom ,gstVector& torques) ;

virtual gstVector calculateForceFieldForce(gstPHANToM* PHANToM) ;
This method will protect current customer's code by allowing it to compile and run in the new scheme where the method is called with two arguments.

double getAttenuationDistance() const;
Get the distance that the force field vector will be attenuated over as the user moves out of the bounding volume containing the force field.

gstForceField* getNextForceFieldInScene() ;
Returns "next" force field in force field scene list.

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 int getStateForPHANToM(gstPHANToM* curPHANToM ,
gstPoint& lastSCP ,
gstVector& exitForce ,
gstVector& lastForce ,

gstVector& exitTorque ,
gstVector& lastTorque) ;

For extension: Return integer representing the contact state for curPHANToM with this shape node. 0 is assumed to be FALSE and represents no contact between curPHANToM and this shape node. Otherwise, the value may indicate special information about the contact between curPHANToM this shape node.

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 the scene graph.

virtual void removeFromSceneGraph() ;
For extension: Used by system or for creating sub-classes only. Called when object is removed from the 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 setAttenuationDistance(const double dist) ;
Sets the distance that the force field force will be attenuated over as the user moves out of the bounding box containing the force field. If this value is zero, the user will feel a buzz when entering a force field in a direction that is opposed by any component of the force vector A good value is in single digit millimeter value range.

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 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.

virtual gstBoolean updateStateForPHANToM(gstPHANToM* curPHANToM ,
int inContact ,
const gstPoint& lastSCP ,
const gstVector& exitForce ,
const gstVector& lastForce ,
const gstVector& exitTorque ,
const gstVector& lastTorque) ;
For extension: Update the contact state between this shape node and curPHANToM. This stores the state for later retrieval with getStateForPHANToM.

static gstVector attenuateToObjectForces(gstVector& forceFieldForce ,
gstVector& objectForces ,
gstPHANToM* phantom) ;
This method is used to attenuate the forces from the force fields to the object forces in the scene.

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

static gstForceField* getForceFieldsInScene() ;
Internal method used to get the number of force fields currently in the scene.

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

Protected constructors

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

gstForceField(const gstForceField& origForceField) ;
Constructor.

gstForceField(const gstForceField* origForceField) ;
Constructor.

Chapter 6. Manipulators

class gstManipulator

Summary

#include “gstManipulator.h”
class gstManipulator ;

Description

Base class for PHANToM manipulator class.

Public constructors

virtual ~gstManipulator() ;
Destructor.

Public Members

virtual gstVector calcManipulatorForce(void* PHANToM ,gstVector& torques) ;

virtual gstVector calcManipulatorForce(void* PHANToM) ;

```
For extension:
Calculate the force.  Force is returned in parent
reference frame of phantom. When subclassing, the first
parameter should be cast to gstPHANToM to retrieve
any information about the state of the PHANToM that
is needed to calculate the forces and move manipulated
node.  Also, if the manipulator is not active, the
zero vector should be returned.
ACTUNG!
WARNING!: Never call PHANToM->setForce or
           PHANToM->setForce_WC from this function.
           It will cause an infinite recursion.
```

gstBoolean getInvertStylusSwitch() const;

gstTransform* getNode() const;
Get node that manipulator modifies.

gstBoolean isActive() const;
Returns TRUE if manip is currently active.

virtual void resetState() ;

```
For extension:
Resets all dynamic state of manipulator.
```

void setInvertStylusSwitch(gstBoolean flag) ;
If TRUE then the inverse value of the stylus switch is used to activate the manipulator. Otherwise, depressing the switch activates the manipulator.

void setNode(gstTransform* newNode) ;
Set "newNode" to be the node in the scene that the manipulator modifies.

void setUseStylusSwitch(gstBoolean _use) ;
Set whether the stylus switch is used to start and stop manipulators. If TRUE, the manipulator is started when

the stylus switch is pressed and remains active until the switch is released. Default is FALSE.

virtual void start() ;
For extension:
Start the manipulator. WARNING: When re-starting a manipulator, make sure to reset any state, such as past PHANToM position. Otherwise, the next call to calcEffectForce could generate unexpectedly large forces.

virtual void stop() ;
For extension:
Stop the effect and reset state.

Protected constructors

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

Protected Data

gstBoolean	active - For extension: TRUE if manipulator is active.
gstBoolean	invertStylusSwitch
gstBoolean	last_active
gstTransform*	manipNode - For extension: Pointer to node to be manipulated.
gstPoint	manipStuckPos
gstBoolean	useStylusSwitch - For extension: TRUE if stylus switch is to be used.

class gstRotateManipulator

Summary

#include "gstRotateManipulator.h"
class gstRotateManipulator : public gstManipulator;

Description

Rotational manipulator. This uses the PHANToM to rotate a node, with force feedback. When active, the manipulator confines the PHANToM to a sphere around the object. The PHANToM also must exceed a threshold force before the manipulator allows the object to be moved. The force feedback is that of a simulated rotating mass with rotational damping.

Public constructors

gstRotateManipulator() ;
Constructor.

~gstRotateManipulator() ;
Destructor.

Public Members

virtual gstVector calcManipulatorForce(void* PHANToMarg ,gstVector& torques) ;

virtual gstVector calcManipulatorForce(void* PHANToMarg) ;

```
For extension:
Calculate the force.  Force is returned in parent
reference frame of phantom. When subclassing, the first
parameter should be cast to gstPHANToM to retrieve
any information about the state of the PHANToM that
is needed to calculate the forces and move manipulated
node.  Also, if the manipulator is not active, the
zero vector should be returned.
ACTUNG!
WARNING!: Never call PHANToM->setForce or
          PHANToM->setForce_WC from this function.
          It will cause an infinite recursion.
```

double getDamping() ;
Get rotational damping of simulated rotating mass [Kg/(1000.0*sec)].

double getMass() const;
Get mass of rotate manipulator [kilograms].

double getSpringK() const;
Get stiffness of spring attached from gstPHANToM to rotating object [Kg/(1000.0*sec^2)].

virtual void resetState() ;
Resets all dynamic state (e.g. Acceleration, velocity) of manipulator.

void setDamping(double _d) ;
Set rotational damping of simulated rotating mass [Kg/(1000.0*sec)].

void setMass(double mass) ;
Set mass of rotate manipulator [kilograms].

	void	setSpringK(double springK) ;

Set stiffness of spring attached from gstPHANToM to rotating object [Kg/(1000.0*sec^2)].

Protected Data

gstTransformMatrix	I - Inertia matrix;
gstTransformMatrix	Iinv - Inverse of I.
gstTransformMatrix	L - Angular momentum.
gstVector	angularVel
gstVector manipVel,	manipAccel
double	manipDamping
double	manipMass
double	manipSpringK
gstQuaternion	q - Rotation quaternion.
gstQuaternion	qL

class gstScaleManipulator

Summary

#include “gstScaleManipulator.h”
class gstScaleManipulator : public gstManipulator;

Description

Scale manipulator. This uses the PHANToM to scale an object, with force feedback. When active, the manipulator generates a force effect as the object is scaled to allow gstPHANToM to scale a node with force feedback. The force effect is modeled as a mass attached to a spring and dashpot in parallel.

Public constructors

gstScaleManipulator() ;
Constructor.

~gstScaleManipulator() ;
Destructor.

Public Members

virtual gstVector calcManipulatorForce(void* PHANToMarg ,gstVector& torques) ;

virtual gstVector calcManipulatorForce(void* PHANToMarg) ;

```
For extension:
Calculate the force.  Force is returned in parent
reference frame of phantom. When subclassing, the first
parameter should be cast to gstPHANToM to retrieve
any information about the state of the PHANToM that
is needed to calculate the forces and move manipulated
node.  Also, if the manipulator is not active, the
zero vector should be returned.
ACTUNG!
WARNING!: Never call PHANToM->setForce or
          PHANToM->setForce_WC from this function.
          It will cause an infinite recursion.
```

double getDamping() const;
Get damping of dashpot attached from gstPHANToM to scaling object [Kg/(1000.0*sec)].

double getMass() const;
Get mass of scale manipulator [kilograms].

double getSpringK() const;
Get stiffness of spring attached from gstPHANToM to scaling object [Kg/(1000.0*sec^2)].

double getThresholdForce() const;
Get threshold force to move object.

virtual void resetState() ;
For extension: Resets all dynamic state (e.g. Acceleration, velocity) of manipulator.

void setDamping(double newDamping) ;
Set damping of dashpot attached from gstPHANToM for scaling object [Kg/(1000.0*sec)].

void setMass(double mass) ;
Set mass of scale manipulator [kilograms].

void setSpringK(double springK) ;
Set stiffness of spring attached from gstPHANToM for scaling object [Kg/(1000.0*sec^2)].

void setThresholdForce(double newThreshold) ;
Set threshold force to move object.

Protected Data

gstPoint	initBoundingRadii
gstVector manipVel,	manipAccel
double	manipDamping
double	manipMass
double	manipScaleThreshold
double	manipSpringK

class gstTranslateManipulator

Summary

#include “gstTranslateManipulator.h”
class gstTranslateManipulator : public gstManipulator;

Description

Translatonal manipulator. This uses the PHANToM to translate an object, with force feedback. When active, the manipulator generates a damping effect as the object is translated. The PHANToM also must exceed a threshold force before the manipulator allows the object to be moved.

Public constructors

gstTranslateManipulator() ;
Constructor.

~gstTranslateManipulator() ;
Destructor.

Public Members

virtual gstVector calcManipulatorForce(void* PHANToMarg ,gstVector& torques) ;

virtual gstVector calcManipulatorForce(void* PHANToMarg) ;

```
For extension:
  Calculate the force.  When subclassing, the first
  parameter should be cast to gstPHANToM to retrieve
  any information about the state of the PHANToM that
  is needed to calculate the forces and move manipulated
  node.  Also, if the manipulator is not active, the
  zero vector should be returned.
  ACTUNG!
  WARNING!: Never call PHANToM->setForce or
            PHANToM->setForce_WC from this function.
            It will cause an infinite recursion.
```

double getManipDamping() const;
Get damping constant of damper attached from gstPHANToM to translating object.

double getMass() const;
Get mass of translate manipulator [kilograms].

double getSpringK() const;
Get stiffness of spring attached from gstPHANToM to translating object.

double getThresholdForce() const;
Get threshold force to move object.

virtual void resetState() ;
For extension: Resets all dynamic state (e.g. Acceleration, velocity) of manipulator.

void setManipDamping(double damping) ;
Set damping constant of damper attached from gstPHANToM to translating object.

void setMass(double mass) ;

Set mass of translate manipulator [kilograms].

void setSpringK(double springK) ;
Set stiffness of spring attached from gstPHANToM to translating object.

void setThresholdForce(double newThreshold) ;
Set threshold force to move object.

Protected Data

gstVector manipVel,	manipAccel
double	manipDamping
double	manipMass
double	manipSpringK
double	manipTranslateThreshold

Chapter 7. Polygon Mesh

class gstTriPoly<gstIncidentEdge, class __default_alloc_template< 1, 0>>

Summary

#include “gstTriPoly.h”
template <gstIncidentEdge, class __default_alloc_template< 1, 0>>
class gstTriPoly : public gstTriPolyBase;

Description

Triangular Poly Class. Every polygon is defined by its 3 vertices in the order specified by the parent gstTriPolyMeshBase class's getVertexOrder method. The parent gstTriPolyMeshBase is specified through the constructor.

Public constructors

gstTriPoly(gstVertex* _v1 ,
gstVertex* _v2 ,
gstVertex* _v3 ,
gstTriPolyMeshBase* mesh = NULL,
const gstPolyKey _key = NULL) ;
Creates Triangle defined by the 3 vertices v1, v2, and v3. V1, v2, and v3 are assumed not to be all colinear and should all be distinct objects. The triangle will have undefined behavior otherwise.

virtual ~gstTriPoly() ;

Public Members

gstBoolean beginModify() ;
Called when 'this' gstTriPoly object is going to be modified.

virtual gstSpatialObject* clone() const;
Virtual method to copy this object and return base class pointer to newly copied object. Allows copy of object to be made using base class pointer.

gstTriPoly* cloneTriPoly() const;
Virtual method to copy this gstTriPoly object and return gstTriPoly pointer to newly cloned object.

gstPoint convertToBarycentricCoord(const gstPoint& pt) const;
Given a 3D point coplaner with this triangle, the corresponding barycentric coordinate is returned. The returned barycentric coordinate is only valid if pt is coplaner and within the boundary of the triangle. Otherwise, the result is undefined.

gstPoint convertToCartesianCoord(const gstPoint2D& uvCoord) const;
Convert UV coordinate to 3D point.

gstPoint2D convertToUVCoord(const gstPoint& pt) const;
Converts 3D point coplaner with triangle to UV coordinates of triangle. U is along v1->v2 vector and V is along v1->v3 vector. Undefined results for points not coplaner with the triangle.

gstEdge* e1() const;
Returns pointer to edge1 (v1->v2).

gstEdge* e2() const;
Returns pointer to edge2 (v2->v3).

gstEdge* e3() const;

Returns pointer to edge3 (v3->v1).

gstBoolean endModify() ;
Called when 'this' gstTriPoly object is done being modified.

virtual gstBoundingBox getBoundingBox() ;
Returns gstBoundingBox just enclosing this triangle.

gstEdge* getE1() const;
Returns pointer to edge1 (v1->v2).

gstEdge* getE2() const;
Returns pointer to edge2 (v2->v3).

gstEdge* getE3() const;
Returns pointer to edge3 (v3->v1).

gstPolyKey getKey() const;
Returns unique id of this object. This id is created at object construction.

gstTriPoly* getPoly(unsigned int index) ;
Returns pointer to gstTriPoly cooresponding to correct index. Index '0' refers to this poly. Indices '1', '2', and '3' refer to the tri poly on the opposite side of edge1, edge2, or edge3 respectively. If no poly exists for the given index NULL is returned.

gstPolyPropertyContainer& getPropertyContainer() ;
Returns pointer to gstPolyPropertyContainer class storing polygonal property information (ie. Vertex normals...) pertaining to this gstTriPoly.

gstTriPolyMeshBase* getTriPolyMesh() const;
Returns pointer to parent gstTriPolyMesh object.

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

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

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

gstVertex* getV3() const;
Returns pointer to v3.

gstBoolean inside(const gstPoint& pt) const;
Returns TRUE if pt is coplaner to this triangle and is within boundary of triangle. If pt is coplaner and outside the boundary of this triangle FALSE is returned. The behavior of this method is undefined if pt is not coplaner with this triangle.

virtual gstVector interpolateNormal(const gstPoint& barycentricCoord) ;
Returns normal vector of Triangle.

gstBoolean intersectCoplanar_LS_PE(const gstLineSegment& lineSeg
,gstLineIntersectionInfoFirstTwo_ParamEdge& intersectionData) ;

LineSeg is a line segment co-planer to this gstTriPoly face. The intersection of the line segment may have 0, 1, or 2 intersections along its length with the vertices or edges of the gstTriPoly. TRUE is returned if there are > 0 intersections. Otherwise FALSE is returned. Information about the intersections is stored in intersectionData.

virtual gstLineIntersectionInfo::IntersectionType intersectFirstInOut_LS_P(const 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 intersectFirstIn_LS_P(const 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 intersectFirstOut_LS_P(const 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 gstObjectIntersectionInfo::IntersectionType intersect_BC(const gstBoundingCube& cube) ;
Intersects gstTriPoly with gstBoundingCube object and returns gstObjectObjectIntersection::intersectionType which can be; RV_OVERLAPPING or RV_NONE. 'RV_OVERLAPPING' specifies that the cube encloses or overlaps the triangle, and 'RV_NONE' specifies that the cube and triangle occupy separate space. This method does not distinguish that the cube completely encloses the triangle.

virtual gstObjectIntersectionInfo::IntersectionType intersect_BS(const gstBoundingSphere& sphere) ;
Intersects gstTriPoly with gstBoundingCube object and returns gstObjectObjectIntersection::intersectionType which can be; RV_OVERLAPPING or RV_NONE. 'RV_OVERLAPPING' specifies that the cube encloses or overlaps the triangle, and 'RV_NONE' specifies that the cube and triangle occupy separate space. This method does not distinguish that the cube completely encloses the triangle.

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

void modify() ;
Called when 'this' gstTriPoly object has been modified.

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 gstTriPolyBase

Summary

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

Description

Base class for triangular polygons.

Public constructors

gstTriPolyBase() ;

virtual ~gstTriPolyBase() ;

Public Members

virtual gstType getTypeId() const;
Get type Id of this instance.

virtual gstBoolean isOfType(gstType type) const;
Returns TRUE if this class is same or derived class of type.

static gstType getClassTypeId() ;
Get type of this class. No instance needed.

static gstBoolean staticIsOfType(gstType type) ;
Returns TRUE if subclass is of type.

class gstTriPolyMesh

Summary

#include “gstTriPolyMesh.h”
class gstTriPolyMesh : public gstTriPolyMeshBase;

Description

Implements collection of triangular polygons that may or may not share vertices and edges.

Public constructors

gstTriPolyMesh() ;
Creates an initially empty set of triangular polygons.

gstTriPolyMesh(const gstTriPolyMesh& mesh) ;
Copy Constructor.

```
gstTriPolyMesh(int numVertices ,
                                        double vertices [ ] [ 3 ] ,
                                        int numTrianglePolygons ,
                                        int trianglePolygons [ ] [ 3 ] ,
                                        gstBoolean useSpatialPartition  = TRUE) ;
```

Constructor. Backward compatible with old gstPolyMesh.

```
gstTriPolyMesh(int numVertices ,
                                        int v_dimension ,
                                        double* vertices ,
                                        int numTrianglePolygons ,
                                        int num_sides ,
                                        int* trianglePolygons ,
                                        gstBoolean useSpatialPartition  = TRUE) ;
```

Constructor.

```
gstTriPolyMesh(int numVertices ,
                                        int v_dimension ,
                                        double** vertices ,
                                        int numTrianglePolygons ,
                                        int num_sides ,
                                        int** trianglePolygons ,
                                        gstBoolean useSpatialPartition  = TRUE) ;
```

Constructor. Backward compatible with old gstPolyMesh.

virtual ~gstTriPolyMesh() ;

Public Members

virtual gstBoolean beginModify(gstSpatialObject* so) ;
Signifies that this object may be modified after this call.

virtual gstSpatialObject* clone() const;
Virtual method to copy this object and return base class pointer to newly copied object. Allows copy of object to be made using base class pointer.

gstTriPolyMesh* cloneTriPolyMesh() const;
Virtual method to copy this gstTriPolyMesh object and return gstTriPolyMesh pointer to newly cloned object.

gstTriPoly* copyPolygon(const gstTriPoly* poly) ;

Create new gstTriPoly by copying poly's vertices first and then creating poly with copied vertices. A pointer to the newly created polygon is returned.

gstVertex* copyVertex(gstVertex* vertToCopy) ;
Creates new gstVertex with same position as vertToCopy if a vertex at that position does not already exist. Also copies the key if the key is not already used by a vertex of this gstTriPolyMesh. A pointer to the already existing coincident vertex or the newly created vertex is returned.

gstTriPoly* createTriPoly(const gstVertexKey v1key ,
const gstVertexKey v2key ,
const gstVertexKey v3key ,
const gstPolyKey polyKey = INT_MAX) ;
Creates new gstTriPoly with vertices v1, v2, and v3 and gives the new polygon a key of polyKey. A pointer to the newly created polygon is returned if successful. If v1, v2, and v3 not valid and unique keys to polygons of this gstTriPolyMesh, then NULL is returned. If polyKey is not unique, then a new unique key is generated and assigned to the new polygon.

gstTriPoly* createTriPoly(gstVertex* v1 ,
gstVertex* v2 ,
gstVertex* v3 ,
gstPolyKey polyKey = INT_MAX) ;
Creates new gstTriPoly with vertices v1, v2, and v3 and gives the new polygon a key of polyKey. A pointer to the newly created polygon is returned if successful. If v1, v2, and v3 are not unique then NULL is returned. If polyKey is not unique, then a new unique key is generated and assigned to the new polygon.

gstVertex* createVertex(const gstPoint vertPos ,const gstVertexKey vertKey = INT_MAX) ;
Create new gstVertex to be used in gstTriPolys that follow. New gstVertex is positioned at vertPos and is indexed by vertkey. If vertKey is not unique, then a unique key and generated and used. A pointer to the newly created gstVertex is returned if successful. Otherwise, NULL is returned.
Note: Does not check for existing vertices of same position or key.

virtual gstBoolean endModify(gstSpatialObject* so) ;
Signifies that modifications to the argument spatial object have ceased. It is assumed that the spatial object is a part of this triangular polygon mesh.

gstVertex* findCoincidentVertex(const gstVertex* vert) ;
Returns pointer to gstVertex coincident with vert if one exists. Otherwise, NULL is returned.

virtual gstBoundingBox getBoundingBox() ;
Returns gstBoundingBox just enclosing this triangle.

virtual gstBoundingSphere getBoundingSphere() ;
Returns gstBoundingSphere just enclosing this triangle.

virtual gstBoolean getContainedObjects(gstSpatialObjectPtrVector& v) ;
Returned vector of pointers to gstSpatialObjects that are contained within this gstTriPolyMesh.
Note: This operation takes proportional time and memory to number of contained objects.

gstBoolean getDeleteStrandedVertices() const;
Returns TRUE if vertices that have no neighbor edges are to be deleted. Otherwise, FALSE is returned.

int getNumPolygons() const;
Returns number of polygons defined for gstTriPolyMesh.

int getNumVertices() const;
Returns number of vertices defined for gstTriPolyMesh. This includes vertices that are stranded (have no neighbor polygons) that have not been deleted.

gstTriPoly* getPolygon(const gstPolyKey polyKey) const;
Returns pointer to gstTriPolygon that has key equal to polyKey if one exists. Otherwise, NULL is returned.

gstSpatialPartition* getSpatialPartition() ;
Returns pointer to gstSpatialPartition defined for this object if one has been created. Otherwise, NULL is returned.

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

gstVertex* getVertex(const gstVertexKey vertKey) const;
Returns pointer to gstVertex that has key equal to vertKey if one exists. Otherwise, NULL is returned.

void initSpatialPartition() ;
Creates a new spatial partition for this object if one hasn't already been defined and initializes it to reflect the mesh defined by this gstTriPolyMesh.

virtual gstLineIntersectionInfo::IntersectionType intersectFirstInOut_LS_P(const 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 intersectFirstInOut_LS_PSO(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_ParamSpatObj& 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 intersectFirstIn_LS_P(const 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 intersectFirstIn_LS_PSO(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_ParamSpatObj& 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 intersectFirstOut_LS_P(const 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 intersectFirstOut_LS_PSO(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_ParamSpatObj& 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 gstObjectIntersectionInfo::IntersectionType intersect_BC(const gstBoundingCube& cube) ;
Not implemented yet.

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

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

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

gstTriPolyPtrHashMapConstIterator polygonsBegin() const;
Returns an iterater pointing to the begining of the polygons defined for this gstTriPolyMesh.

gstTriPolyPtrHashMapConstIterator polygonsEnd() const;
Returns an iterater pointing to the end of the polygons defined for this gstTriPolyMesh.

gstBoolean removePolygon(const gstPolyKey polyKey) ;
Removes gstTriPoly defined for this mesh with key equal to polyKey and returns TRUE if one exists that meets this criteria. Otherwise, FALSE is returned.

gstBoolean removePolygon(gstTriPoly** polyToRemove) ;
Removes gstTriPoly polyToRemove if it is contained by this mesh and returns TRUE. Otherwise, FALSE is returned.

gstBoolean removeVertex(const gstVertexKey vertKey) ;
Removes gstVertex defined for this mesh with key equal to vertKey and returns TRUE if one exists that meets this criteria. Otherwise, FALSE is returned.

void setDeleteStrandedVertices(const gstBoolean deleteStrandedVertices) ;
If deleteStrandedVertices is TRUE, vertices that have no neighbor edges are to be deleted. Otherwise, they are left alone.

void setSpatialPartition(gstSpatialPartition* s) ;
Sets spatial partition defined for this object to s.

void setUseSpatialPartition(gstBoolean u) ;
If u is TRUE the spatial partition defined for this mesh is used to speed up calculations if one exists. Otherwise, all polygons are examined for calculations requiring it.

gstBoolean useSpatialPartition() const;
Returns TRUE if the spatial partition defined for this mesh is used to speed up calculations if one has been created. Otherwise FALSE is returned meaning all polygons are examined for calculations requiring it.

gstVertexPtrHashMapConstIterator verticesBegin() const;
Returns begining iterator for vertices.

gstVertexPtrHashMapConstIterator verticesEnd() const;
Returns ending iterator for vertices.

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 gstTriPolyMeshBase

Summary

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

Description

Base class for triangular polygons class.

Enums

enum **VertexOrder_**
Enumeration used to specify clockwise or counterclockwise specification of polygon normal based on vertices.
RV_CLOCKWISE
RV_COUNTERCLOCKWISE

Public constructors

virtual ~gstTriPolyMeshBase() ;

Public Members

virtual gstBoolean beginModify(gstSpatialObject*) ;
Signifies that this object may be modified after this call.

virtual void endModifications() ;
Signifies that modifications to this object and any gstTriPoly objects making up the triangular mesh have ceased.

virtual gstBoolean endModify(gstSpatialObject*) ;
Signifies that modifications to the argument spatial object have ceased. It is assumed that the spatial object is a part of this triangular polygon mesh.

double getCreaseAngle() const;
Return the angle tolerance between faces.

gstModifyBase* getOwner() const;
Returns the owner object of derived type gstModifyBase that uses or owns this object.

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

gstTriPolyMeshBase::VertexOrder getVertexOrder() ;
Set vertex order triangle polygons are specified by to determine their surface normal. This value can be RV_CLOCKWISE or RV_COUNTERCLOCKWISE.

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

void modify(gstTriPoly* modifiedTriPoly) ;
Signifies that modifiedTriPoly has been modified.

void setCreaseAngle(double a) ;
Set the angle tolerance between faces.

void setOwner(gstModifyBase* owner) ;
Specifies an object of derived type gstModifyBase that uses or owns this object. The owner object will have it's modified method called when this object is modified.

void setVertexOrder(gstTriPolyMeshBase::VertexOrder newOrder) ;

Set vertex order triangle polygons are specified in to determine their surface normal. This value can be RV_CLOCKWISE, RV_COUNTERCLOCKWISE.

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

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

gstTriPolyMeshBase(const gstTriPolyMeshBase& mesh) ;
Copy constructor.

class gstTriPolyMeshHaptic

Summary

#include “gstTriPolyMeshHaptic.h”
class gstTriPolyMeshHaptic : public gstShape, public gstModifyBase;

Description

Triangle polygon scene graph node. Represents a triangular polygon mesh that is haptically palpable.

Enums

enum **TouchableFrom_**
RV_FRONT
RV_BACK
RV_FRONT_AND_BACK
enum **resultValues**
RV_NONE
RV_FACE
RV_EDGE
RV_CORNER

Public constructors

gstTriPolyMeshHaptic() ;
Defined new gstTriPolysH object with empty polygon mesh.

gstTriPolyMeshHaptic(const gstTriPolyMeshHaptic& mesh) ;

gstTriPolyMeshHaptic(gstTriPolyMesh* triMesh) ;

virtual ~gstTriPolyMeshHaptic() ;

Public Members

virtual gstNode* Clone() const;
Virtual method to copy this object and return base class pointer to newly copied object. Allows copy of object to be made using base class pointer.

gstTriPolyMeshHaptic* CloneTriPolyMeshHaptic() const;
Virtual method to copy this gstTriPolyMeshHaptic object and return gstTriPolyMeshHaptic pointer to newly cloned object.

virtual gstBoolean collisionDetect(gstPHANToM* phantomNode) ;
For extension: Used by system or for creating sub-classes only. Returns TRUE if the 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().

gstBoolean getSmoothing() const;
Returns TRUE if normal interpolation smoothing is to be used.
Note: Normal polygon property must be specified at vertices for this to take effect.

gstTriPolyMeshHaptic::TouchableFrom getTouchableFrom() const;
Returns RV_FRONT, RV_BACK, or RV_FRONT_AND_BACK to denote which sides of the triangles will be touchable by the PHANToM.

const gstTriPolyPtrVector& getTouchedPolys() const;
Returns a vector of pointers to gstTriPoly. This vector will contain 0, 1, 2, or 3 values if there was no, face, edge, or corner contact respectively. For edge and corner contact the 2nd and 3rd values indicate the polys defining the edge and corner respectively.

gstTriPolyMeshBase* getTriPolyMesh() const;
Returns a pointer to the gstTriPolyMesh storing the polygonal mesh for this object. If none has been set yet then NULL is returned.

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

virtual gstLineIntersectionInfo::IntersectionType intersectCacheFirstInOut_LS_PSO(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_ParamSpatObj& 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 intersectCacheFirstIn_LS_PSO(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_ParamSpatObj& 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 intersectCacheFirstOut_LS_PSO(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_ParamSpatObj& 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 intersectFirstInOut_LS_PSO(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_ParamSpatObj& 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 intersectFirstIn_LS_PSO(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_ParamSpatObj& 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 intersectFirstOut_LS_PSO(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_ParamSpatObj& 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 gstBoolean intersect_LS_PN(gstLineSegment& lineSegment
,gstLineIntersectionInfoFirst_ParamNormal& intersectionInfo) ;

virtual gstBoolean intersect_LS_PN_WC(gstLineSegment& lineSegment_WC
,gstLineIntersectionInfoFirst_ParamNormal& intersectionInfo_WC) ;

virtual gstLineIntersectionInfo::IntersectionType intersect_LS_PSO(const gstLineSegment& lineSegment
,gstLineIntersectionInfoFirst_ParamSpatObj& 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 gstBoolean intersection(const gstPoint& startPt_WC ,
const gstPoint& endPt_WC ,
gstPoint& intersectionPt_WC ,
gstVector& intersectionNormal_WC ,
void** data) ;
For extension: Used by system or for creating sub-classes only. Returns TRUE if the line segment, defined by startPt_WC and endPt_WC in the world coordinate system, intersects the shape object. If TRUE, intersectionPt_WC is set to the point of intersection and intersectionNormal_WC is set to the surface normal at the intersection point.

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

virtual void modified() ;
Signifies that modifications to this object have occured and any calculated state must be updated.

virtual void scale(const gstPoint& _scale) ;

virtual void scale(double s) ;

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

virtual void setCenter(const gstPoint& _center) ;

virtual void setScale(const gstPoint& _scale) ;

virtual void setScale(double scale) ;

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

void setSmoothing(gstBoolean smoothing) ;
If smoothing is TRUE normal interpolation smoothing is to be used.
Note: Normal polygon property must be specified at vertices for this to take effect.

void setTouchableFrom(gstTriPolyMeshHaptic::TouchableFrom touchableFrom)
;
Sets touchable sides of triangles to value of touchableFrom. The possible values are RV_FRONT, RV_BACK, or RV_FRONT_AND_BACK.

void setTriPolyMesh(gstTriPolyMeshBase* newTriMesh) ;
Sets newTriMesh as the gstTriPolyMesh storing the polygonal mesh for this object.

void updateBoundingShape() ;
Updates boundingObject based on current state of polygonal mesh.

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.

Additional Classes

class gstBT_GeomObj

Summary

#include “gstBinTree.h”
class gstBT_GeomObj ;

Description

Wrapper class for gstSpatialObjects so that they all have nice generic properties to be handled by the partition class.

Public constructors

gstBT_GeomObj(gstSpatialObject* o) ;

~gstBT_GeomObj() ;

Public Members

void dec() ;

void inc() ;

gstPoint& maxPt() ;
Access maximum point of bounding box.

gstPoint& minPt() ;
Access minimum point of bounding box.

gstSpatialObject* obj() ;
Access contained object.

class gstBT_Node

Summary

#include “gstBinTree.h”
class gstBT_Node ;

Description

This class represents nodes within the gstBinTree class.

Public constructors

gstBT_Node() ;

~gstBT_Node() ;

Public Members

int& axis() ;
Active axis access method.

int& depth() ;
Depth of this node.

gstBT_Node* getChild(int i) ;
Get the ith child node. Note unsafe behavior.

void getChildren(int axis ,
const gstPoint& origin ,
gstBT_Node** nearP ,
gstBT_Node** farP) ;
Sort both children into a "near" and "far" division relative to a particular axis and plane.

gstBT_Node* getParent() ;
Get the parent node.

gstBinTree* getTree() ;
Get the partition tree which contains this node.

gstPoint& maxPt() ;
Access minimum extent point.

gstBT_GeomObjList& members() ;
List access methods.

gstPoint& minPt() ;
Access minimum extent point.

void setChild(int i ,gstBT_Node* child) ;
Set the ith child node. Note unsafe behavior.

void setParent(gstBT_Node* p) ;
Set the parent node.

void setTree(gstBinTree* t) ;
Set the partition tree which contains this node.

class gstBT_Stack

Summary

#include “gstBinTree.h”
class gstBT_Stack ;

Description

Class to linearize the traversal of the gstBinTree. WARNING! This class does NOT check to see if it is exceeding its depth bounds. This is for speed. Be careful.

Public constructors

gstBT_Stack(int depth = 50) ;
Ctor. Initializes and allocates memory for stack.

~gstBT_Stack() ;
Dtor. Deletes objects in stack and deletes stack itself.

Public Members

int empty() ;
Is the stack empty?

void init() ;
Initialize the stack by emptying it.

gstBT_Node* pop(double& min ,double& max) ;
Pop top node off of stack, returning node and bounds.

void push(gstBT_Node* node ,
double min ,
double max) ;
Push a new node and bounds onto the stack.

class gstBT_StackElem<gstBT_GeomObj*, class __default_alloc_template< 1, 0>>

Summary

#include “gstBinTree.h”
template <gstBT_GeomObj*, class __default_alloc_template< 1, 0>>
class gstBT_StackElem ;

Description

Class to represent an element of the partition traversal stack.

Public constructors

gstBT_StackElem() ;

~gstBT_StackElem() ;

class gstBinTree

Summary

#include “gstBinTree.h”
class gstBinTree : public gstSpatialPartition;

Description

A particular type of spatial partition. This is a binary tree almost as described in various parts of the Graphics Gems series. With minor modifications and bug fixes.

Enums

enum **Type**
Type of intersection.
In - Intersection going in.
Out - Intersection going out.
Either - Either direction.

Public constructors

gstBinTree(gstSpatialObject* so) ;

virtual ~gstBinTree() ;

Public Members

gstLineIntersectionInfo::IntersectionType BT_intersect_LS_P(const gstLineSegment& lineSegment , gstLineIntersectionInfoFirst_Param& intersectionInfo , gstBinTree::Type type) ;
Interface for gstLineIntersectionInfoFirst_Param intersection methods.

gstLineIntersectionInfo::IntersectionType BT_intersect_LS_PSO(const gstLineSegment& lineSegment , gstLineIntersectionInfoFirst_ParamSpatObj& intersectionInfo , gstBinTree::Type type) ;
Interface for gstLineIntersectionInfoFirst_ParamSpatObj intersection methods.

void addGeomToNode(gstBT_GeomObj* obj ,gstBT_Node* node) ;
Add an object to a node.

gstBoolean bboxContained(const gstBoundingBox& bbox) ;
Does the partition box contain any of the box passed in.

gstBoolean bboxIntersect(const gstBoundingBox& bbox ,gstBT_NodeSet& nodes) ;
Retrieve those objs whose bboxes intersect this bbox.

virtual gstBoolean beginModify(gstSpatialObject* so) ;
Remove object from tree to prepare for modification.

gstBoolean boxIntersect(const gstLineSegment& lseg , const gstVector& lseg_unit_dir , const gstPoint& min , const gstPoint& max , double& returnMin , double& returnMax) ;
Low level intersection bounding routine.

virtual gstBoolean endModify(gstSpatialObject* so) ;

Reinsert object into tree after modification.

virtual gstBoundingBox getBoundingBox() ;
Return union of bounding boxes of contained objects. Cached.

virtual gstBoundingSphere getBoundingSphere() ;
Return bounding sphere of bounding box.

gstBoundingBox getHardBox() const;
Get the hardwaired partition space.

gstPoint getMax() const;
Return the maximum point of the partition space.

gstPoint getMin() const;
Return the minimum point of the partition space.

gstSpatialObject* getParent() ;
Method to access parent poly collection.

double getScale() const;
Get the overall partition space scale factor.

gstBT_Stack* getStack() ;
Get the internal traversal stack.

gstBoolean getUseHardBox() const;
Get whether to use a hardwired partition space.

virtual void init() ;
Initialize the tree.

void initStack() ;
Initialize the internal traversal stack.

void insertGeomIntoNode(gstBT_GeomObj* obj ,gstBT_Node* node) ;
Insert this object into this node, subdividing if necessary.

gstBoolean insert_object(gstSpatialObject* so) ;
Insert this gstSpatialObject into the tree.

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

virtual gstLineIntersectionInfo::IntersectionType intersectFirstInOut_LS_PSO(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_ParamSpatObj& intersectionInfo) ;
See gstSpatialObject.

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

virtual gstLineIntersectionInfo::IntersectionType intersectFirstIn_LS_PSO(const gstLineSegment&

lineSegment ,gstLineIntersectionInfoFirst_ParamSpatObj& intersectionInfo) ;
See gstSpatialObject.

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

virtual gstLineIntersectionInfo::IntersectionType intersectFirstOut_LS_PSO(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_ParamSpatObj& intersectionInfo) ;
See gstSpatialObject.

virtual gstObjectIntersectionInfo::IntersectionType intersect_BC(const gstBoundingCube& cube) ;
GstBoundingCube intersection methods.

gstBoolean isGeomInNode(gstBT_GeomObj* obj ,gstBT_Node* node) ;
Test if an object is in a node.

gstBT_GeomObjList& members() ;
Return the object list.

gstBoolean objIntersect(const gstLineSegment& lseg ,
double min ,
double max ,
gstBT_GeomObjList& objList ,
gstBT_GeomObj** obj ,
gstLineIntersectionInfoFirst_Param& intersectionInfo ,
gstBinTree::Type type) ;
Low level object collection intersection routine.

void popStack(gstBT_Node** node ,
double& min ,
double& max) ;
Pop a node off the stack returning the newest top item.

void pushStack(gstBT_Node* node ,
double min ,
double max) ;
Push a node onto the stack.

void removeGeomFromNode(gstBT_GeomObj* obj ,gstBT_Node* node) ;
Remove an object from a node.

gstBT_Node* root() ;
Access to child nodes methods.

void setHardBox(gstBoundingBox b) ;
Set the hardwaired partition space.

void setMax(const gstPoint& m) ;
Set the maximum point of the partition space.

void setMin(const gstPoint& m) ;
Set the minimum point of the partition space.

void setScale(double s) ;
Set the overall partition space scale factor.

void setUseHardBox(gstBoolean u) ;
Set whether to use a hardwired partition space.

gstBoolean treeIntersect(const gstLineSegment& lseg ,
gstBT_GeomObj** obj ,
gstLineIntersectionInfoFirst_Param& intersectionInfo ,
gstBinTree::Type type) ;
The principal work horse intersection function.

void wipe() ;
Void the tree.

static double getEpsilon() ;
Get the precision of the tree.

static int getMaxListLength() ;
Get the maximum number of objects in a leaf node of the tree.

static int getMaxTreeDepth() ;
Get the maximum tree depth for the class.

static double getPartitionOverlapTol() ;
Get the overlap between node bounding boxes as a percentage of the box size.

static void setEpsilon(double e) ;
Get the precision of the tree.

static void setMaxListLength(int m) ;
Get the maximum number of objects in a leaf node of the tree.

static void setMaxTreeDepth(int m) ;
Get the maximum tree depth for the class.

static void setPartitionOverlapTol(double t) ;
Get the overlap between node bounding boxes as a percentage of the box size.

Protected members

void bbox_intersect(const gstBoundingBox& bbox ,
gstBT_NodeSet& nodes ,
gstBT_Node* node) ;
Recursive bounding box intersection routine used by bboxIntersect.

gstSpatialObjectSet& getModifySet() ;
Get the set of objects currently being modified.

void subdivide(gstBT_Node* node) ;

class gstLineIntersectionInfo

Summary

#include “gstLineIntersectionInfo.h”
class gstLineIntersectionInfo ;

Description

Stores status of intersection of line with some object. This status may indicate no intersection, no intersection with line inside object, no intersection with line outside object, intersection into object, intersection into and out of object, intersection out of object, and intersection through object using the respective enumerations of gstLineIntersectionInfo::IntersectionType.

Enums

enum **resultValues**
RV_NONE
RV_NONE_INSIDE
RV_NONE_OUTSIDE
RV_IN
RV_INOUT
RV_OUT
RV_THROUGH

Public constructors

gstLineIntersectionInfo() ;
Constructor.

Public Members

gstLineIntersectionInfo::IntersectionType getStatus() const;
Returns the status of the intersection.

void setStatus(gstLineIntersectionInfo::IntersectionType status) ;
Sets status of the intersection to status.

gstLineIntersectionInfo::IntersectionType status() const;
Returns the status of the intersection.

Public Data

IntersectionType d_status

class gstLineIntersectionInfoFirstTwo_Param

Summary

#include “gstLineIntersectionInfoFirstTwo_Param.h”
class gstLineIntersectionInfoFirstTwo_Param : public gstLineIntersectionInfoFirst_Param;

Description

First two intersections along line with object including parametric values (t1 and t2) of line where first two intersections occur.

Public Members

double getT2() const;
Returns value of T2.

void setT2(const double t2) ;
Sets value of T2 to t2.

double t2() const;
Returns value of T2.

class gstLineIntersectionInfoFirstTwo_ParamEdge

Summary

#include “gstLineIntersectionInfoFirstTwo_ParamEdge.h”
class gstLineIntersectionInfoFirstTwo_ParamEdge : public gstLineIntersectionInfoFirstTwo_Param;

Description

First two intersections along line coplanar with polygonal face including parametric values (t1 and t2) of line where first two intersections occur and includes pointers to the polygon's edges (e1 and e2) for the respective two intersections.

Public Members

gstEdge* e1() const;
Returns value of E1.

gstEdge* e2() const;
Returns value of E2.

gstEdge* getE1() const;
Returns value of E1.

gstEdge* getE2() const;
Returns value of E2.

void setE1(gstEdge* e1) ;
Sets value of E1 to e1.

void setE2(gstEdge* e2) ;
Sets value of E2 to e2.

class gstLineIntersectionInfoFirst_Param

Summary

#include “gstLineIntersectionInfoFirst_Param.h”
class gstLineIntersectionInfoFirst_Param : public gstLineIntersectionInfo;

Description

First intersection along line with object including parametric value (t1) of line where first intersection occurs.

Public constructors

virtual ~gstLineIntersectionInfoFirst_Param() ;

Public Members

double getT1() const;
Returns value of T1.

void setT1(const double t1) ;
Sets value of T1 to t1.

double t1() const;
Returns value of T1.

class gstLineIntersectionInfoFirst_ParamNormal

Summary

#include “gstLineIntersectionInfoFirst_ParamNormal.h”
class gstLineIntersectionInfoFirst_ParamNormal : public gstLineIntersectionInfoFirst_Param;

Description

First intersection along line with object including parametric value (t1) of line where first intersection occurs and normal of surface at point of intersection.

Public Members

const gstVector& getNormal() const;
Returns value of intersection normal.

const gstVector& normal() const;
Returns value of intersection normal.

void setNormal(const gstVector& normal) ;
Sets value of intersection normal to normal.

class gstLineIntersectionInfoFirst_ParamSpatObj

Summary

#include “gstLineIntersectionInfoFirst_ParamSpatObj.h”
class gstLineIntersectionInfoFirst_ParamSpatObj : public gstLineIntersectionInfoFirst_Param;

Description

First intersection along line with object including parametric value (t1) of line where first intersection occurs and pointer to gstSpatialObject intersected by line.

Public constructors

gstLineIntersectionInfoFirst_ParamSpatObj() ;
Constructor.

virtual ~gstLineIntersectionInfoFirst_ParamSpatObj() ;

Public Members

gstSpatialObject* getSpatialObject() const;
Returns pointer to intersected spatial object.

void setSpatialObject(gstSpatialObject* spatialObj) ;
Sets pointer to intersected spatial object to spatialObjs.

gstSpatialObject* spatialObject() const;
Returns pointer to intersected spatial object.

class gstLineIntersectionInfoFirst_ParamTriPoly

Summary

#include “gstLineIntersectionInfoFirst_ParamTriPoly.h”
class gstLineIntersectionInfoFirst_ParamTriPoly : public gstLineIntersectionInfoFirst_Param;

Description

First intersection along line with object including parametric value (t1) of line where first intersection occurs and pointer to gstTriPoly intersected by line.

Public constructors

virtual ~gstLineIntersectionInfoFirst_ParamTriPoly() ;

Public Members

gstTriPoly* getTriPoly() const;
Returns pointer to intersected gstTriPoly.

void setTriPoly(gstTriPoly* triPoly) ;
Sets pointer of intersected gstTriPoly to triPoly.

gstTriPoly* triPoly() const;
Returns pointer to intersected gstTriPoly.

class gstModifyBase

Summary

#include “gstModifyBase.h”
class gstModifyBase ;

Description

Interface to signal when a instance is modified.

Public Members

virtual void modified() ;

class gstNormalPolyProperty

Summary

#include “gstNormalPolyProperty.h”
class gstNormalPolyProperty : public gstPolyPropertyPoint3D;

Description

For assigning normals to vertices in a gstPolyMesh.

Public constructors

gstNormalPolyProperty() ;

gstNormalPolyProperty(const gstPoint& val) ;

Public Members

virtual int	Id() ;
static int	staticGetId() ;

Protected constructors

virtual	~gstNormalPolyProperty() ;

class gstObjectIntersectionInfo

Summary #include “gstObjectIntersectionInfo.h”
class gstObjectIntersectionInfo ;

Description Structure for various intersection tests.

Enums enum **resultValues**
RV_NONE
RV_OVERLAPPING
RV_ENCLOSED
RV_ENCLOSING

Public Data

int	test

class gstOnePolyPropertyIdStruct

Summary

#include “gstPolyPropertyContainer.h”
class gstOnePolyPropertyIdStruct ;

Description

One poly property with an ID.

Public Operators

gstBoolean	operator<(const gstOnePolyPropertyIdStruct& e) const;
Less than operator.	
gstBoolean	operator==(const gstOnePolyPropertyIdStruct& e) const;
Equality test operator.	

Public Data

int	Id
gstPolyPropertyBase*	prop

class gstPolyPropertyBase

Summary

#include “gstPolyPropertyBase.h”
class gstPolyPropertyBase ;

Description

Base class for property associated with an entire polygon or one of its vertices (ie. Normal vector). Properties may be 1,2, or 3 dimensional quantities stored and returned as double, gstPoint2D, and gstPoint respectively. The base class does not implement storage for any data, but does provide virtual methods to access data. All data accessors return FALSE in the base class. Derived classes are to override the proper accessors to allow access to the data.

Public constructors

virtual ~gstPolyPropertyBase() ;
The destructor is kept protected since this is a reference counted object.

Public Members

virtual int Id() = 0;
Pure virtual method to be implemented by derived classes. This method should return a unique integer Id for the class. This Id should be created as a static member of the derived class using the createNewPropertyId method of this class.

virtual gstBoolean getDoubleValue(double& val) const;
Accessor to property data of type double. This method currently returns FALSE and has no effect on val. If a derived class stores double property data, then this class is to return TRUE and place the double value into val.

virtual gstBoolean getPoint2DValue(gstPoint2D& val) const;
Accessor to property data of type gstPoint2D. This method currently returns FALSE and has no effect on val. If a derived class stores gstPoint2D property data, then this class is to return TRUE and place the gstPoint2D value into val.

virtual gstBoolean getPoint3DValue(gstPoint& val) const;
Accessor to property data of type gstPoint. This method currently returns FALSE and has no effect on val. If a derived class stores gstPoint property data, then this class is to return TRUE and place the gstPoint value into val.

```
virtual gstBoolean interpolateBarycentric(gstPolyPropertyBase* prop1 ,
                                          gstPolyPropertyBase* prop2 ,
                                          gstPolyPropertyBase* prop3 ,
                                          const gstPoint& barycentricCoord ,
                                          double& result ) const;
```

Given three gstPolyPropertyBase derived objects, prop1, prop2, and prop3, that are assumed to store double data, the double 'result' value is calculated by interpolating the 3 property values based on the barycentric coordinate value barycentricCoord. If successful, TRUE is returned. Otherwise, FALSE is returned.

```
virtual gstBoolean interpolateBarycentric(gstPolyPropertyBase* prop1 ,
                                          gstPolyPropertyBase* prop2 ,
                                          gstPolyPropertyBase* prop3 ,
                                          const gstPoint& barycentricCoord ,
                                          gstPoint& result ) const;
```

Given three gstPolyPropertyBase derived objects, prop1, prop2, and prop3, that are assumed to store gstPoint data, the gstPoint 'result' value is calculated by interpolating the 3 property values based on the barycentric coordinate value barycentricCoord. If successful, TRUE is returned. Otherwise, FALSE is returned.

virtual gstBoolean interpolateBarycentric(gstPolyPropertyBase* prop1 ,
gstPolyPropertyBase* prop2 ,
gstPolyPropertyBase* prop3 ,
const gstPoint& barycentricCoord ,
gstPoint2D& result) const;
Given three gstPolyPropertyBase derived objects, prop1, prop2, and prop3, that are assumed to store gstPoint2D data, the gstPoint2D 'result' value is calculated by interpolating the 3 property values based on the barycentric coordinate value barycentricCoord. If successful, TRUE is returned. Otherwise, FALSE is returned.

virtual gstBoolean isDoubleProp() const;
Returns TRUE if the property stores double data.

virtual gstBoolean isPoint2DProp() const;
Returns TRUE if the property stores gstPoint2D data.

virtual gstBoolean isPoint3DProp() const;
Returns TRUE if the property stores gstPoint data.

void ref() ;
Denotes that some object is storing a pointer to this object by incrementing the reference counter. Once the pointer is no longer stored by that object, unref() should be called.

virtual gstBoolean setDoubleValue(const double& val) ;
Returns TRUE if the property stores double data and stores the value val. Otherwise, FALSE is returned.

virtual gstBoolean setPoint2DValue(const gstPoint2D& val) ;
Returns TRUE if the property stores gstPoint2D data and stores the value val. Otherwise, FALSE is returned.

virtual gstBoolean setPoint3DValue(const gstPoint& val) ;
Returns TRUE if the property stores gstPoint data and stores the value val. Otherwise, FALSE is returned.

void unref() ;
Denotes that an object previously storing a pointer to this object is no longer doing so by decrementing the reference counter for this object. If the reference counter is decremented to a value less than or equal to zero then the storage for this object is deleted.

Protected constructors

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

Protected members

static int createNewPropertyId() ;
This method generates a unique integer Id that is mean't to be used as a unique Id for each derived class of gstPolyPropertyBase.

class gstPolyPropertyContainer<gstThreePolyPropertyIdStruct, class __default_alloc_template< 1, 0>>

Summary

#include “gstPolyPropertyContainer.h”
template <gstThreePolyPropertyIdStruct, class __default_alloc_template< 1, 0>>
class gstPolyPropertyContainer ;

Description

Container class to hold indexed property information of 1,2, and/or 3 dimensional quality pertaining to an entire polygon or to it's individual vertices.

Public constructors

virtual ~gstPolyPropertyContainer() ;

Public Members

gstBoolean evalDoubleProperty(int polyPropertyId ,
gstPoint barycentricCoord ,
double& val) ;
Evaluates the value of the gstPolyProperty with Id() equal to polyPropertyId at the barycentric coordinate barycentricCoord and stores the value in 'val'. Returns TRUE if successful. If one value of polyPropertyId is stored through the setPropertyPoly method then 'val' is set to this and TRUE is returned. If three values of polyPropertyId are stored for vertices v1, v2, and v3 then 'val' is set to the interpolated value at barycentricCoord and TRUE is returned. If no property of Id equal to polyPropertyId is stored then FALSE is returned.

gstBoolean evalPoint2DProperty(int polyPropertyId ,
gstPoint barycentricCoord ,
gstPoint2D& val) ;
Evaluates the value of the gstPolyProperty with Id() equal to polyPropertyId at the barycentric coordinate barycentricCoord and stores the value in 'val'. Returns TRUE if successful. If one value of polyPropertyId is stored through the setPropertyPoly method then 'val' is set to this and TRUE is returned. If three values of polyPropertyId are stored for vertices v1, v2, and v3 then 'val' is set to the interpolated value at barycentricCoord and TRUE is returned. If no property of Id equal to polyPropertyId is stored then FALSE is returned.

gstBoolean evalPoint3DProperty(int polyPropertyId ,
gstPoint barycentricCoord ,
gstPoint& val) ;
Evaluates the value of the gstPolyProperty with Id() equal to polyPropertyId at the barycentric coordinate barycentricCoord and stores the value in 'val'. Returns TRUE if successful. If one value of polyPropertyId is stored through the setPropertyPoly method then 'val' is set to this and TRUE is returned. If three values of polyPropertyId are stored for vertices v1, v2, and v3 then 'val' is set to the interpolated value at barycentricCoord and TRUE is returned. If no property of Id equal to polyPropertyId is stored then FALSE is returned.

gstPolyPropertyBase* getPropertyPoly(int polyPropertyId) ;
Returns a pointer to the property associated with the entire poly with Id equal to polyPropertyId. If none exists, NULL is returned.

gstPolyPropertyBase* getPropertyV1(int polyPropertyId) ;
Returns a pointer to the property associated with vertex v1 with Id equal to polyPropertyId. If none exists, NULL is returned.

gstPolyPropertyBase* getPropertyV2(int polyPropertyId) ;

Returns a pointer to the property associated with vertex v2 with Id equal to polyPropertyId. If none exists, NULL is returned.

gstPolyPropertyBase* getPropertyV3(int polyPropertyId) ;
Returns a pointer to the property associated with vertex v3 with Id equal to polyPropertyId. If none exists, NULL is returned.

gstBoolean removePropertyPoly(int polyPropertyId) ;
Removes the property associated with the entire poly with Id equal to polyPropertyId and returns TRUE. If none exists, FALSE is returned.

gstBoolean removePropertyVerts(int polyPropertyId) ;
Removes the property associated with vertices with Id equal to polyPropertyId and returns TRUE. If none exists, FALSE is returned.

gstBoolean setPropertyPoly(gstPolyPropertyBase* prop) ;
Stores the property prop and associates it with the entire poly and returns TRUE. Returns FALSE is prop is NULL.

gstBoolean setPropertyV1(gstPolyPropertyBase* prop) ;
Stores the property prop and associates it with vertex v1 of poly and returns TRUE. Returns FALSE is prop is NULL.

gstBoolean setPropertyV2(gstPolyPropertyBase* prop) ;
Stores the property prop and associates it with vertex v2 of poly and returns TRUE. Returns FALSE is prop is NULL.

gstBoolean setPropertyV3(gstPolyPropertyBase* prop) ;
Stores the property prop and associates it with vertex v3 of poly and returns TRUE. Returns FALSE is prop is NULL.

Protected members

gstThreePolyPropertyIdStruct* getVertPropertyStruct(int polyPropertyId) ;
Returns structure for vertex properties of Id = polyPropertyId.

class gstPolyPropertyDouble

Summary

#include “gstPolyPropertyDouble.h”
class gstPolyPropertyDouble : public gstPolyPropertyBase;

Description

Property storing double data associated with an entire polygon or one of its vertices (ie. Normal vector).

Public constructors

gstPolyPropertyDouble() ;

gstPolyPropertyDouble(const double& val) ;

Public Members

virtual gstBoolean getDoubleValue(double& val) const;
Accessor to property data of type double. Returns TRUE and places the double value into val.

virtual gstBoolean isDoubleProp() const;
Returns TRUE to identify that this property does store values of type double.

virtual gstBoolean setDoubleValue(const double& val) ;
Returns TRUE and stores val.

Protected constructors

virtual ~gstPolyPropertyDouble() ;

class gstPolyPropertyPoint2D

Summary

#include “gstPolyPropertyPoint2D.h”
class gstPolyPropertyPoint2D : public gstPolyPropertyBase;

Description

Property storing gstPoint2D data associated with an entire polygon or one of its vertices (ie. Normal vector).

Public constructors

gstPolyPropertyPoint2D() ;

gstPolyPropertyPoint2D(const gstPoint2D& val) ;

Public Members

virtual gstBoolean getPoint2DValue(gstPoint2D& val) const;
Accessor to property data of type gstPoint2D. Returns TRUE and places the gstPoint2D value into val.

virtual gstBoolean isPoint2DProp() const;
Returns TRUE to identify that this property does store values of type gstPoint2D.

virtual gstBoolean setPoint2DValue(const gstPoint2D& val) ;
Returns TRUE and stores val.

Protected constructors

virtual ~gstPolyPropertyPoint2D() ;

class gstPolyPropertyPoint3D

Summary

#include “gstPolyPropertyPoint3D.h”
class gstPolyPropertyPoint3D : public gstPolyPropertyBase;

Description

Property storing gstPoint data associated with an entire polygon or one of its vertices (ie. Normal vector).

Public constructors

gstPolyPropertyPoint3D() ;

gstPolyPropertyPoint3D(const gstPoint& val) ;

Public Members

virtual gstBoolean getPoint3DValue(gstPoint& val) const;
Accessor to property data of type gstPoint. Returns TRUE and places the gstPoint value into val.

virtual gstBoolean isPoint3DProp() const;
Returns TRUE to identify that this property does store values of type gstPoint.

virtual gstBoolean setPoint3DValue(const gstPoint& val) ;
Returns TRUE and stores val.

Protected constructors

virtual ~gstPolyPropertyPoint3D() ;

class gstSpatialObject<gstSpatialObject*, class __default_alloc_template< 1, 0>>

Summary

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

Description

Base class intended for all GHOST objects of spatial extent. This class allows access to functionality common to such classes.

Public constructors

virtual ~gstSpatialObject() ;

Public Members

virtual gstSpatialObject* clone() const;
Return a copy of this object as a gstSpatialObject.

gstSpatialObject* cloneSpatialObject() const;
Return a copy of this object as a gstSpatialObject.

virtual gstBoundingBox getBoundingBox() ;
Return the bounding box of the object.

virtual gstBoundingSphere getBoundingSphere() ;
Return the bounding sphere of the object.

virtual gstBoolean getContainedObjects(gstSpatialObjectPtrVector&) ;
Gather the collection of objects contained within this object. By default this returns FALSE indicating that there is no collection.

virtual gstType getTypeId() const;
Get type of this instance.

virtual gstLineIntersectionInfo::IntersectionType intersectFirstInOut_LS_P(const 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 intersectFirstInOut_LS_PSO(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_ParamSpatObj& 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 intersectFirstInOut_Ray_P(const 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.

virtual gstLineIntersectionInfo::IntersectionType intersectFirstInOut_Ray_PSO(const gstRay& ray ,gstLineIntersectionInfoFirst_ParamSpatObj& 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 intersectFirstIn_LS_P(const 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 intersectFirstIn_LS_PSO(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_ParamSpatObj& 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 intersectFirstIn_Ray_P(const 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.

virtual gstLineIntersectionInfo::IntersectionType intersectFirstIn_Ray_PSO(const gstRay& ray ,gstLineIntersectionInfoFirst_ParamSpatObj& 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 intersectFirstOut_LS_P(const 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 intersectFirstOut_LS_PSO(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_ParamSpatObj& 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 intersectFirstOut_Ray_P(const 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.

virtual gstLineIntersectionInfo::IntersectionType intersectFirstOut_Ray_PSO(const gstRay& ray ,gstLineIntersectionInfoFirst_ParamSpatObj& 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 gstObjectIntersectionInfo::IntersectionType intersect_BC(const gstBoundingCube& cube) ;
Intersects gstSimpleCube with spatial object and returns gstIntersection::intersectionType which can be; enclosed, enclosing, overlapping, or none. 'enclosed' specifies that the simple cube is enclosed within the object, 'enclosing' specifies that the simple cube encloses the spatial object, 'overlapping' specifies that the simple cube and the spatial object overlap eachother, and 'none' specifies that the simple cube and spatial object occupy separate space.

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

virtual gstLineIntersectionInfo::IntersectionType intersect_LS_P(const 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_LS_PSO(const gstLineSegment& lineSegment ,gstLineIntersectionInfoFirst_ParamSpatObj& 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_Ray_P(const 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.

virtual gstLineIntersectionInfo::IntersectionType intersect_Ray_PSO(const gstRay& ray ,gstLineIntersectionInfoFirst_ParamSpatObj& 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 gstBoolean isOfType(gstType type) const;
Returns TRUE this class is same or derived class of type.

static gstType getClassTypeId() ;
Get type of this class. No instance needed.

static gstBoolean staticIsOfType(gstType type) ;
Returns TRUE if subclass is of type.

Protected constructors

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

class gstSpatialPartition

Summary

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

Description

Class to act as a base class for all types of spatial partitions which will get imposed on GHOST objects.

Public constructors

gstSpatialPartition() ;

virtual ~gstSpatialPartition() ;

Public Members

virtual gstBoolean beginModify(gstSpatialObject*) ;
Notification method indicating that a particular gstSpatialObject inside of the partition is going to be modified/moved. Should return value indicating success or failure in finding the specified gstSpatialObject inside of the partition. Base class version does nothing.

virtual gstBoolean endModify(gstSpatialObject*) ;
Notification method indicating that a particular gstSpatialObject inside of the partition is finished being modified/moved. Should return value indicating success or failure in finding the specified gstSpatialObject inside of the partition. Base class version does nothing.

virtual void init() ;
Triggers the building of the partition. This does not happen by default at construction time.

class gstThreePolyPropertyIdStruct

Summary

#include “gstPolyPropertyContainer.h”
class gstThreePolyPropertyIdStruct ;

Description

Three poly properties with an ID.

Public Operators

gstBoolean	operator<(const gstThreePolyPropertyIdStruct& e) const;

Less than operator.

gstBoolean	operator==(const gstThreePolyPropertyIdStruct& e) const;

Equality test operator.

Public Data

int	Id
gstPolyPropertyBase*	prop1
gstPolyPropertyBase*	prop2
gstPolyPropertyBase*	prop3

Function operator<<

Summary #include "gstLineIntersectionInfo.h"

```
ostream& operator<<(ostream& os ,const
gstLineIntersectionInfo::IntersectionType& type ) ;
```

Description Insert operator for IntersectionType.

Function __declspec

Summary #include "gstObjectIntersectionInfo_SpatialObjectPtrSet.h"
```
_STD_END class __declspec(dllimport )
gstObjectIntersectionInfo_SpatialObjectPtrSet;
```

Chapter 8. Misc

class gstTimer

Summary

#include “gstTimer.h”
class gstTimer ;

Description

Wraper for measuring elapsed time.

Public constructors

gstTimer() ;

virtual ~gstTimer() ;

Public Members

void	reset() ;
float	secondsSinceLastQuery() ;
float	secondsSinceReset() ;

class gstTimerRecord

Summary

#include “gstTimer.h”
class gstTimerRecord ;

Public constructors

gstTimerRecord(const char* recordName ,int maxUpdates = MAX_TIMER_UPDATES) ;

virtual ~gstTimerRecord() ;

Public Members

void outputTimerData() ;
Stop the timer and store delta.

void start() ;

void stop() ;
Start the timer.

static void setFileName(const char* fileName) ;
Handles writing all of the presently logged data to file.

Function IsRecoverableError

Summary #include "gstErrorHandler.h"

```
gstBoolean IsRecoverableError(int errorId ) ;
```

Description Check whether the error is recoverable.

Function debugTimerStart

Summary #include "gstTimer.h"
```
void debugTimerStart() ;
```

Description Old debugTimer convenience method.

Function debugTimerStop

Summary #include "gstTimer.h"
```
float debugTimerStop() ;
```

Description Old debugTimer convenience method.

Function getErrorMessage

Summary

#include "gstErrorHandler.h"

```
char* getErrorMessage(int errorId ) ;
```

Description

Retrieve generic error message string.

Function gstErrorHandler

Summary #include "gstErrorHandler.h"

```
void gstErrorHandler(int errorId ,char* quote ) ;
```

Description Generic error handler. Prints error message to stderror.

Function gstErrorHandler

Summary

#include "gstErrorHandler.h"

```
void gstErrorHandler(int errorId ,
                     char* quote ,
                     double errorValue ) ;
```

Description Generic error handler. Prints error message to stderror.

Summary

#include "gstErrorHandler.h"

```
void gstErrorHandler(int errorId ,
                     char* quote ,
                     gstPoint errorValue ) ;
```

Description Generic error handler. Prints error message to stderror.

Function gstErrorHandler

Summary

#include "gstErrorHandler.h"

```
void gstErrorHandler(int errorId ,
                     char* quote ,
                     gstVector errorValue ) ;
```

Description Generic error handler. Prints error message to stderror.

Summary

#include "gstErrorHandler.h"

```
void gstErrorHandler(int errorId ,
                     char* quote ,
                     int errorValue ) ;
```

Description

Generic error handler. Prints error message to stderror.

Function gstErrorHandler

Summary

#include "gstErrorHandler.h"

```
void gstErrorHandler(int errorId ,
                     char* quote ,
                     unsigned int errorValue ) ;
```

Description Generic error handler. Prints error message to stderror.

Summary

#include "gstErrorHandler.h"

```
void gstErrorHandler(int errorId ,
                     char* quote ,
                     void* errorValue ) ;
```

Description Generic error handler. Prints error message to stderror.

Function gstGetPDDVersion

Summary

#include "gstBasic.h"

```
const char* gstGetPDDVersion() ;
```

Description

Returns PDD version string.

Function gstGetVersion

Summary #include "gstBasic.h"

```
const char* gstGetVersion() ;
```

Description Returns GHOST version string.

Function gstSpew

Summary

#include "gstSpew.h"

```
void gstSpew(gstSeparator* sep  = 0,
                        ostream& os  = cout,
                        int level  = 0) ;
```

Description

Print scene graph out as text for debugging.

Function printErrorMessages

Summary #include "gstErrorHandler.h"

```
void printErrorMessages(gstBoolean flag ) ;
```

Description If set to TRUE then error messages will be automatically printed. Otherwise, user must print error messages using errorCallback.

Function setErrorCallback

Summary #include "gstErrorHandler.h"

```
void setErrorCallback(gstErrorCallback* newCallback ,void* userdata ) ;
```

Description Set user error callback. This callback is called for each error. The parameters passed in are the error number, character string description of error, and pointer to user data.

Summary

#include "gstBasic.h"

```
gstBoolean withinEpsilon(double d1 ,
                         double d2 ,
                         double epsilon  = 0. 00000001) ;
```

Description

Check if two numbers are (nearly) the same.

Section II - GHOST GL library

class gfxDisplaySettings

Summary

#include “ghostGLManager.h”
class gfxDisplaySettings ;

Description

Per node display lists and settings.

Public constructors

gfxDisplaySettings(ghostGLManager* gfxManager_ = NULL) ;

Public Members

static void setDefaultGeometryLock(gstBoolean bLock) ;
Used for storing userdata for callback.

Public Data

static gstBoolean	bDefaultGeometryLock
gstBoolean	bGeometryLocked - When enabled, this can lead to significant speed increase.
gstGraphicsCallback*	callback - Used for storing userdefined callback.
int	geomDisplayList - Compiled OpenGL display list that renders this node's geometry.
ghostGLManager*	gfxManager - GhostGLManager for this node.
int	postDisplayList - Unset material properties, texture mapping, etc.
int	preDisplayList - Set material properties, texture mapping, etc.
void*	userdata - Used for storing userdata for callback.
int	xformDisplayList - Compiled OpenGL display list that manipulates the ModelView matrix stack.

class gfxPhantomDisplaySettings

Summary #include “ghostGLManager.h”
class gfxPhantomDisplaySettings : public gfxDisplaySettings;

Description Display lists and settings for the gstPHANToM node.

Public constructors gfxPhantomDisplaySettings(ghostGLManager* m) ;

Public Data

gstBoolean	bDrawSCP

class ghostGLActionObject

Summary

#include “ghostGLActionObject.h”
class ghostGLActionObject ;

Description

Base class for action objects such as the pinch transform.

Public constructors

virtual ~ghostGLActionObject() ;

Public Members

virtual void remove() ;
Remove is called when ghostGLManager::removeActionObject() is called on this object.

Protected constructors

ghostGLActionObject() ;
Constructor is protected because it is not meaningful to instantiate a ghostGLActionObject, only objects derived from it.

Protected members

ghostGLManager* getGLManager() ;
Accessor.

virtual void postDisplay() ;
Called after each display iteration.

virtual void preDisplay() ;
Called before each display iteration.

void setGLManager(ghostGLManager* glManager) ;
Assign new manager.

Protected Data

ghostGLManager* m_glManager - Pointer to our manager.

class ghostGLCameraBase

Summary

#include “ghostGLCameraBase.h”
class ghostGLCameraBase ;

Description

A flexible OpenGL interface for a basic camera viewing transform. The transform can either be set explicitly or determined by setting camera properties, such as: position, look at, view angle, etc.

Note: Changing a property about the camera will automatically cause an update on the internally maintained transform matrix. It is advisable to disable updates when modifying multiple properties and then force an update when you're done.

Public constructors

ghostGLCameraBase() ;

virtual ~ghostGLCameraBase() ;

Public Members

virtual void applyTransform() ;
Draws the current camera transform onto the OpenGL matrix stack.

double getFarClippingPlane() const;

ghostGLManager* getGLManager() const;

gstPoint getLookAt() const;

double getNearClippingPlane() const;

gstPoint getPosition() const;
Gets the current camera position in world coordinates and returns a const gstPoint reference.

gstTransformMatrix getTransformMatrix(gstBoolean bInverseForm = FALSE) ;
Returns the current transform matrix being used for drawing the camera. If bInverseForm is set to TRUE, then the actual inverse form used for drawing the camera against the OpenGL matrix stack is returned.

gstPoint getUpVector() const;

double getViewAngle() const;

int getWindowHeight() const;

int getWindowWidth() const;
Getter methods for the window dimensions.

virtual void loadCameraTransform() ;
Uses the current camera position, look at and up vector settings to determine the camera transform matrix.

virtual void reshape(int width ,int height) ;
Update the OpenGL viewport and perspective settings as a result of a change in width, height, view angle, or clipping planes.

virtual void setClippingPlanes(const double nearClip ,
const double farClip ,

gstBoolean bUpdate = TRUE) ;
Accessor methods for the camera's near / far Z clipping planes.

void setGLManager(ghostGLManager* pGLManager) ;
Accessor methods for the camera's glManager instance.

virtual void setLookAt(const gstPoint& point ,gstBoolean bUpdate = TRUE) ;
Accessor methods for camera's LookAt.

virtual void setPosition(const gstPoint& position ,gstBoolean bUpdate = TRUE) ;
Accessor methods for the camera's position.

virtual void setTransformMatrix(gstTransformMatrix& cameraXform ,
gstBoolean bInverseForm = FALSE,
gstBoolean bUpdate = TRUE) ;
Accessor methods for the camera's transform matrix.

virtual void setUpVector(const gstVector& upVec ,gstBoolean bUpdate = TRUE) ;
Accessor methods for camera's up vector.

virtual void setViewAngle(const double viewAngle ,gstBoolean bUpdate = TRUE) ;
Accessor methods for the camera's view angle (field of view).

virtual void update() ;
Performs transform and view updates depending on whether a particular camera property changed.
Note: If you choose to delay an update until all the camera settings are made, this routine will determine whether to update the camera transform and/or reshape the view.

Protected members

virtual void initDefault() ;
Initializes the camera to its default state.

```
Position = (0, 0, 350)
Look At = (0, 0, 0)
Up Vector = (0, 1, 0)
View Angle = 40 degrees
Near Clipping Plane = 1
Far Clipping Plane = 1000
```

Protected Data

static const double kDefaultCameraX - Default X position.
static const double kDefaultCameraY - Default Y position.
static const double kDefaultCameraZ - Default Z position.
static const double kDefaultFarClip - Default far Z clipping plane.
static const double kDefaultFov - Default camera field of view.
static const double kDefaultNearClip - Default near Z clipping plane.
gstBoolean m_bReshapeView
gstBoolean m_bUpdateTransform - These flags are used to enable updates when parameters change.
double m_farClip
gstPoint m_lookAt
double m_nearClip
ghostGLManager* m_pGLManager
gstTransformMatrix m_transform

gstVector	m_upVec
double	m_viewAngle
int	m_windowHeight
int	m_windowWidth - Updated whenever a reshape event occurs.

class ghostGLDraw

Summary

#include “ghostGLDraw.h”
class ghostGLDraw ;

Description

Provides OpenGL based routines for drawing GHOST nodes. Each primitive has a generic drawing routine based on its geometric description as well as a GHOST wrapper. Generally, you'll want to pass a GHOST object to the corresponding wrapper function, which will handle calling the appropriate OpenGL drawing routine with the necessary parameters.

The GHOST wrapper routines all take two parameters. The first parameter is a pointer to the gstShape object. The second parameter specifies whether you want ghostGLDraw to apply the object's local transform to the OpenGL ModelView matrix stack before drawing the geometry.

Public constructors

ghostGLDraw() ;

virtual ~ghostGLDraw() ;

Public Members

static void drawBoundaryCube(gstBoundaryCube* cubeH ,gstBoolean bDrawTransform = TRUE) ;

static void drawBoundaryCube(const double length ,
const double height ,
const double width) ;

static void drawCone(const double radius ,const double height) ;

static void drawCone(gstCone* coneH ,gstBoolean bDrawTransform = TRUE) ;

static void drawCoordAxes(double scale = kDefaultCoordAxisScale) ;

static void drawCube(gstCube* cubeH ,gstBoolean bDrawTransform = TRUE) ;

static void drawCube(const double length ,
const double height ,
const double width) ;

static void drawCylinder(const double radius ,const double height) ;

static void drawCylinder(gstCylinder* cylinderH ,gstBoolean bDrawTransform = TRUE) ;

static void drawNode(gstTransform* transformNode ,gstBoolean bDrawTransform = TRUE) ;

static void drawPhantom(gstPHANToM* phantom ,gstBoolean bDrawTransform = TRUE) ;

static void drawPoint(const gstPoint& pt) ;

static void drawSeparator(gstSeparator* sep ,gstBoolean bDrawTransform = TRUE) ;
Draws separator transform. Does NOT draw children.

	static void	drawSphere(const double radius) ;
	static void	drawSphere(gstSphere* sphereH ,gstBoolean bDrawTransform = TRUE) ;
	static void	drawTorus(double inner ,double outter) ;
	static void	drawTorus(gstTorus* torus ,gstBoolean bDrawTransform = TRUE) ;
	static void	drawTransform(gstTransform* transformObj) ;
	static void	drawTransformMatrix(gstTransformMatrix& matrixH) ;
	static void	drawTriPolyMesh(gstTriPolyMesh* triMesh) ;
	static void	drawTriPolyMeshHaptic(gstTriPolyMeshHaptic* triMeshH ,gstBoolean bDrawTransform = TRUE) ;
	static void	drawTriangle(gstTriPoly* tri) ;
	static void	drawVector(const gstVector& vec) ;
Protected Data	static GLUquadricObj*	m_pQuadObj

class ghostGLManager

Summary

#include “ghostGLManager.h”
class ghostGLManager : public ghostGLDraw;

Description

The GhostGL Manager class makes up the core of the GhostGL library. It is primarily responsible for managing a scene graph that has been populated with GHOST primitives. In doing so, it sets up the necessary graphics callbacks and manages OpenGL display lists for drawing each node in a scene.

When the glManager loads a scene, it creates a gfxDisplaySettings structure for each node in the scene. This structure holds a series of display lists that get called whenever a node is drawn. By default, all gstShape primitives in the scene will have geometry display lists generated for them by using the drawing routines in ghostGLDraw. After a scene has been loaded, you can access the gfxDisplaySettings for any node in the scene to change rendering properties, or even swap in your own geometry display list for rendering.

This implementation has not been tested for use with multiple views of a scene.

Public constructors

ghostGLManager(ghostGLCameraBase* camera = NULL) ;
Sets up the camera and initializes OpenGL state for the rendering context.

virtual ~ghostGLManager() ;
Cleans up the objects belonging to this ghostGLManager instance.
Note : If you created your own camera, you'll need to delete it yourself.

Public Members

virtual void addActionObject(ghostGLActionObject* obj) ;
Adds an action object to be pinged every redraw. This allows you to have dynamic operations occuring at the graphics rate.

void clearAll() ;
Cleanup methods that remove the gfxDisplaySettings and callback related data.

void clearNode(gstTransform* node) ;
Frees the display lists and other GhostGL data associated with this node.
Note : If you have your own user-defined callback and data, clearing the node will still preserve your callback and data.

void clearSeparator(gstSeparator* sep) ;
Calls clearNode() on all descendents of the separator.

ghostGLCameraBase* getCamera() const;

gfxDisplaySettings* getDisplaySettings(gstTransform* node) ;
Returns the gfxDisplaySettings instance for a particular node.

gstTransformMatrix getPhantomCumulativeTransform() const;

gfxPhantomDisplaySettings* getPhantomDisplaySettings(gstPHANToM* phantom) ;
Returns the gfxPhantomDisplaySettings node for a gstPHANToM node.

gstPHANToM* getPhantomNode() const;
Returns the first PHANToM instance in the pool of current PHANToMs.

gstTransformMatrix getPhantomTransform() const;
Get data about the PHANToM from the last graphics callback.

gstScene* getScene() ;

void* getUserData(gstTransform* node) ;
Returns a previously set userdata element.

gstBoolean isAxesOn() const;

gstBoolean isGraphicsOn() const;

virtual void loadScene(gstScene* scene) ;
Handles setting up the gstScene instance for GhostGL rendering. Each node in the scene needs to be setup with a graphics callback and a gfxDisplaySettings structure.

virtual void redraw() ;
The main entry point for having ghostGL draw the scene. It handles calling all of the action objects, applying the current camera transform and the uses drawScene() to render all of the nodes.

virtual void removeActionObject(ghostGLActionObject* obj) ;
Removes the specified action object instance from the list of action objects.

virtual void reshape(int width ,int height) ;
Reshape should be called with the current view dimensions everytime the rendering context is resized.

void setAxesOn(gstBoolean bEnable) ;
Accessor methods for turning on/off the coordinate origin.

virtual void setCamera(ghostGLCameraBase* camera) ;
Specify your own camera for viewing the scene.

void setGeometryLock(gstTransform* node ,gstBoolean bLock) ;
If you know that a nonstatic node is not going to be changing its geometry frequently, you can lock the geometry and get better rendering performance.
This routine will work recursively off of a separator, or you can just perform this on an individual node.

void setGraphicsCallback(gstTransform* node ,
gstGraphicsCallback* callback ,
void* userdata = NULL) ;
Since ghostGLManager uses GHOST's own graphics callback fields to store information, it provides its own callback/userdata field. This method sets a callback and userdata pair for a given node. The graphics callback will be called from ghostGLManager's own graphics callback. NULL for either argument will set that field to NULL. There is no "keep previous" if you only want to change one argument.

void setGraphicsOn(gstBoolean bEnable) ;
Accessor methods for enabling/disabling drawing.

void setTouchableByPHANToM(gstBoolean bTouchable) ;
Change properties about the scene in a local or global manner.

void setVisible(gstTransform* node ,gstBoolean bVisible) ;
Allows you to recursively specify visiblity for one or several nodes. If a separator node is passed in, all of its descendants will be affected. When a node is made invisible, all of its display list entries are deleted. When the

node is made visible again, it will automatically be setup again for rendering.

static void updateNode(gstTransform* node ,
void* cbData ,
void* userData) ;
This is the graphics callback routine used for all nodes except the PHANToM.

static void updatePhantom(gstTransform* node ,
void* cbData ,
void* userData) ;
This is the graphics callback routine used for just PHANToM nodes.

Protected members

void callDisplayLists(gstTransform* node ,ghostGLManager::ghostGLDrawMode drawMode = ALL_DISPLAY_LISTS) ;
Handles calling the display lists for the passed in node. If the draw mode is ALL_DISPLAY_LISTS, then the display lists are called in this order: preDisplayList, xformDisplayList, geomDisplayList, postDisplayList. If the draw mode is PRE_DISPLAY, then only the first three display lists are called. The POST_DISPLAY mode will only call the postDisplayList.

void callNode(gstSeparator* node) ;
Recursively calls the display lists for all descendants of a separator. If the node to be display is not a separator, then all of the display lists are called one after the other. If the node is a separator, then the PRE_DISPLAY mode is used before calling all of the descendants. After all descendant have been called, the POST_DISPLAY will be called for the separator. This allows you to set a rendering property in the pre/post display list entries for a separator and affect a whole branch in the scene.

virtual void drawScene() ;
Method used for rendering the scene during each redraw.

virtual void initEnv() ;
Initializes the OpenGL rendering state for backface culling, depth buffering, lighting, and smooth shading.

virtual void setupNode(gstTransform* transformNode ,gstTransformGraphicsCBData* cbData) ;
This routine is used for generating the display list entries for a node. If a node hasn't been setup for ghostGL rendering, a gfxDisplaySettings instance will be created for the node. Each time afterwards, the xformDisplayList entry will be updated and even the geomDisplayList if the geometry hasn't been locked.

virtual void setupPhantom(gstPHANToM* phantomNode ,gstPHANToMGraphicsCBData* cbData) ;
This routine is used for generating the display list entries for a gstPHANToM node. If the node hasn't been setup for ghostGL rendering, an instance of gfxPhantomDisplaySettings will be created for the node. Each time afterwards, the xformDisplayList entry will be updated. The geometry for a PHANToM node is locked by default.

virtual void setupSeparator(gstSeparator* node) ;
Methods used for setting up nodes for GhostGL rendering.

Protected Data

list< ghostGLActionObject*> m_actionObjs - The list of action objects.

class ghostGLPinchXForm

Summary

#include “ghostGLPinchXForm.h”
class ghostGLPinchXForm : public ghostGLActionObject;

Description

Allow the PHANToM position/orientation to change the camera view. The effect is as if clicking the stylus button grabbed/pinched to scene and allowed you to manipulate these scene with the PHANToM. But note that the camera is actually moving, the scene is not.

Public constructors

ghostGLPinchXForm() ;
Constructor.

virtual ~ghostGLPinchXForm() ;

Public Members

virtual void preDisplay() ;
Checks if we should start or stop.

virtual void remove() ;
Stop us before we get removed. Remove is called when ghostGLManager::removeActionObject() is called on this object.

Protected members

void start() ;
Enable pinch mode.

void stop() ;
End pinch mode.

void updatePinchXform() ;
Call each graphics update while pinch is enabled.

class ghostGLSyncCamera

Summary

#include “ghostGLSyncCamera.h”
class ghostGLSyncCamera : public ghostGLCameraBase;

Description

Automatically determines an optimal viewing transform and/or haptic orientation transform. This camera offers two principal modes of operation that get performed whenever a reshape or update occurs.

By default, the sync camera operates in SYNC_WORKSPACE_TO_CAMERA mode, which works as a synchronization mechanism for setting the PHANToM base transform to align with the current camera transform. (i.e. What you feel is aligned with what you see).

NOTE : You must have a distinct parent separator above the PHANToM for this mechanism to work. Just having a root separator isn't enough.

```
                  geometry
root separator <
                  PHANToM parent <
                                    gstPHANToM instance
```

The second mode of operation is called SYNC_CAMERA_TO_WORKSPACE. This method requires that you use a gstBoundaryCube to define a haptic boundary on your PHANToM. (Your scene graph requires a special configuration for this to work. The easiest way to set this up is to call the method attachMaximalBoundary() on your gstPHANToM instance. This method will attach a gstBoundaryCube as a sibling of the PHANToM in the scene.

NOTE : You must have a distinct parent separator above the PHANToM for this mechanism to work. Just having a root separator isn't enough.

```
                  geometry
root separator <                    gstBoundaryCube
                  PHANToM parent <
                                    gstPHANToM instance
```

You will also need to explicitly set the workspace boundary that you want to use with the sync camera by calling setWorkspaceBoundary().

Once this setup is complete, the camera will take care of modifying its transform and viewing parameters to see the entirety of the workspace. If you wanted to modify how the PHANToM is oriented in the scene, you can change the transform of the PHANToM's parent and the camera will adapt to the new workspace transform. This mode allows you to "see what you feel".

There is also a third useful option called SyncToWindow that works along with both of the previously mentioned sync modes. This option will adapt the workspace dimensions in X and Y to maximally fit the aspect ratio of your viewport. This option is enabled by default.

Enums

enum **CameraSyncMode**
SYNC_WORKSPACE_TO_CAMERA
SYNC_CAMERA_TO_WORKSPACE

Public constructors

ghostGLSyncCamera() ;
Sets some default parameters for camera synchronization.

virtual ~ghostGLSyncCamera() ;

Public Members

gstBoolean getClipToWorkspace() const;

CameraSyncMode getSyncMode() const;

gstBoolean getSyncToWindow() const;

gstBoolean getViewEntireWorkspace() const;

gstBoundaryCube* getWorkspaceBoundary() const;

gstPoint getWorkspaceOffset() const;

gstBoolean isSyncEnabled() const;

virtual void reshape(int width ,int height) ;
A reshape of the viewport triggers a SyncToWindow update.

void setClipToWorkspace(gstBoolean bEnable) ;
Accessor methods for OpenGL z-clipping of the workspace.

void setSyncEnabled(gstBoolean bEnable) ;
Allows you to enable/disable camera syncing altogether.

void setSyncMode(ghostGLSyncCamera::CameraSyncMode syncMode) ;
Accessor methods for setting the principal camera sync mode.

void setSyncToWindow(gstBoolean bEnable) ;
Accessor methods for the SyncToWindow state.

void setViewEntireWorkspace(gstBoolean bEnable) ;
Accessor methods for front plane or back plane syncing.

void setWorkspaceBoundary(gstBoundaryCube* boundary) ;
Accessor methods for the workspace boundary.

void setWorkspaceOffset(gstPoint& offset) ;
Accessor methods for workspace offset used by SYNC_WORKSPACE_TO_CAMERA.

virtual void update() ;
Performs an update on the camera base and then synchronizes the camera via an update on the camera transform and/or PHANToM base transform.

Protected members

gstPHANToM* getPhantomNode() ;
Returns the gstPHANToM instance associated with the glManager.

virtual float setViewParameters() ;
Method used for the SyncToWindow option.

virtual void syncCameraToWorkspace() ;
Method used for SYNC_WORKSPACE_TO_CAMERA.

virtual void syncWorkspaceToCamera() ;
This sync method is used by the SYNC_WORKSPACE_TO_CAMERA mode. It uses the current workspace

offset setting to pretranslate the PHANToM's base relative to the camera transform. The main utility of this sync method is that it ensures the haptics are aligned with the camera view (i.e. What you see is what you feel).

virtual void syncWorkspaceToWindow() ;
Method used for SYNC_CAMERA_TO_WORKSPACE.

Protected Data

gstBoolean m_bClipToWorkspace - Set the zClipping to clip to the workspace front and back.
gstBoolean m_bSyncEnabled - Enable/disable all syncing activity.
gstBoolean m_bSyncToWindow - Enable/disable syncing of the workspace dimensions to the viewport's aspect ratio.
gstBoolean m_bViewEntireWorkspace - Controls whether the camera is positioned to view the entirety of the workspace or just the back plane.
gstBoundaryCube* m_pBoundary - The phantom workspace boundary to view.
CameraSyncMode m_syncMode - The mode of synchronization being used.
gstVector m_workspaceOffset - The offset between the camera and the workspace.

class ghostGLUTManager

Summary

#include “ghostGLUTManager.h”
class ghostGLUTManager : public ghostGLManager;

Description

Adds windowing with GLUT to ghostGLManager. With ghostGLUTManager only a few lines of code can create a fully functional graphical-haptic simulation.

Public Members

virtual void redraw() ;

void startMainloop() ;

static ghostGLUTManager* CreateInstance(int argc ,
char* argv [ ] ,
char* title = DEFAULT_WINDOW_TITLE) ;
Creates the instance of the singleton class. If a singleton factory object has been registered (see SetFactory()), This function calls the factory object to create the singleton instance. If no factory object is registered, this function directly creates a singleton instance.

static ghostGLUTManager* GetInstance() ;
Returns a pointer to the singleton instance if one has been created. Otherwise NULL is returned.

static bool HasInstance() ;
Call this function to determine if a singleton instance currently exists. This function returns TRUE (non-zero) a singleton instance does exist.

static ghostGLUTManager* PeekInstance() ;
Call this function to directly obtain a pointer to the singleton instance. If no such instance exists, this function returns zero without creating a new instance.
CAUTION:
This is a low-level function. In general you should use GetInstance() instead of this function. This function is provided for effeciency and should be use with caution.

static bool ReleaseInstance(bool bAutoFree = FALSE) ;
Releases a pointer to the singleton object previously obtained by calling GetInstance().
The function returns TRUE if a singleton instance still exists upon return from this function.

static void glutDisplay() ;

static void glutIdle() ;

static void glutReshape(int width ,int height) ;
Automatically free unused resources.

Protected constructors

ghostGLUTManager() ;
Protected default constructor prevents direct creation of instances of this class. Actual instance creation is peformed by the CreateInstance() member function.

virtual ~ghostGLUTManager() ;
Protected destructor prevents direct deletion of the singleton instance. Actual instance deletion is performed by the DestroyInstance() member function.

Protected members

void init(int argc ,
char* argv [ ] ,
char* title = DEFAULT_WINDOW_TITLE) ;

static void DestroyInstance() ;
Destroys the singleton instance (if such an instance exists).

Section III - GHOST VRML Reader

class gstVRMLError

Summary

#include “gstVRMLError.h”
class gstVRMLError ;

Description

Class to handle VRML file reading errors. Records the error type, file string related to error, and the line number where the error was encountered.

Public constructors

gstVRMLError() ;

gstVRMLError(const gstVRMLError& err) ;

~gstVRMLError() ;

Public Operators

gstVRMLError& operator=(const gstVRMLError& err) ;

Public Members

gstVRMLErrorType GetError() const;
Return the specific error type.

int GetLine() const;
Return the line number which was current when error encountered.

const char* GetMSG() const;
Return the last parsed text string when error encountered.

Function gstReadVRMLFile

Summary

#include "gstVRML.h"

```
gstSeparator* gstReadVRMLFile(const char* filename ) ;
```

Description

Read a VRML 2 file and convert into a GHOST scene graph. Given the name of a file in the VRML 2.0 format, this function returns a gstSeparator containing the VRML scene graph in the GHOST v2 format.

Function gstVRMLClearErrors

Summary

#include "gstVRML.h"

```
void gstVRMLClearErrors() ;
```

Description

Clear all errors.

Function gstVRMLGetEarliestError

Summary

#include "gstVRML.h"

```
gstVRMLError gstVRMLGetEarliestError() ;
```

Description

Get earliest error.

Function gstVRMLGetError

Summary

#include "gstVRML.h"

```
gstVRMLError gstVRMLGetError() ;
```

Description

Get earliest error, return no error if none.

Function gstVRMLGetErrorTypeName

Summary #include "gstVRML.h"

```
const char* gstVRMLGetErrorTypeName(gstVRMLErrorType errType ) ;
```

Description Convert error type to string representation.

Function gstVRMLGetLatestError

Summary

#include "gstVRML.h"

```
gstVRMLError gstVRMLGetLatestError() ;
```

Description

Get latest error.

Function gstVRMLGetNumErrors

Summary #include "gstVRML.h"

```
int gstVRMLGetNumErrors() ;
```

Description How many errors.

Function gstVRMLPopEarliestError

Summary

#include "gstVRML.h"

```
gstVRMLError gstVRMLPopEarliestError() ;
```

Description

Get earliest and remove it from list.

Function gstVRMLPopLatestError

Summary

#include "gstVRML.h"

```
gstVRMLError gstVRMLPopLatestError() ;
```

Description

Get latest and remove it from list.

Function gstVRMLPushError

Summary

```
#include "gstVRMLError.h"
void gstVRMLPushError(gstVRMLErrorType error ,
                      int line ,
                      const char* msg ) ;
```

Function gstVRMLWriteErrorsToFile

Summary #include "gstVRML.h"

```
int gstVRMLWriteErrorsToFile(const char* filename ) ;
```

Description Dump errors to given filename.

enum gstVRMLErrorType

Summary #include "gstVRMLError.h"

Description The possible error types during the GHOST VRML reading process.

Values **gst_VET_NoError** - There was no error.

gst_VET_NoFile - The file was not found.

gst_VET_CannotAddNullNode - An attempt to add a null node.

gst_VET_CurrentNodeIsNull - An attempt to add a node to a null node.

gst_VET_RootNodeIsNull - There is no valid root node.

gst_VET_CouldNotFindNodeByName - A node specified by name could not be found.

gst_VET_CouldNotFindParentNode - A node that should have a parent could not locate it.

gst_VET_AttemptToPushNullNode - An attempt to add a null node.

gst_VET_FieldConversionFailure - Text string could not be converted to proper type.

gst_VET_NonUniformScaleOnNonLeaf - Attempt to perform nonuniform scale on leaf node.

gst_VET_ScaleOrientationNotSupported

gst_VET_UnknownNodeType - A node that is of some unrecognized type.

gst_VET_UnknownFieldType - A field that is of some unrecognized type.

gst_VET_SyntaxError - Encountered a syntax error during parsing.

gst_VET_InvalidHeader

Index

Please Press F9 for the Index.