Shapoid

P. Baillehache

January 15, 2018

Contents

Introduction

Shapoid is a C library providing the `Shapoid` structure and its functions which can be used to manipulate Shapoid objects (see next section for details).

It uses the `PBErr`, `PBMath` and `GSet` library.

1 Definitions

A Shapoid is a geometry defined by its dimension $D \in \mathbb{N}_{+}^{*}$ equals to the number of dimensions of the space it exists in, its position $\overrightarrow{P}$, and its axis $(\overrightarrow{A_0}, \overrightarrow{A_1}, ..., \overrightarrow{A_{D-1}})$. A_i and P are vectors of dimension D. In what follows I'll use I as notation for the interval $[0, D-1]$ for simplification.

Shapoids are classified in three groups: Facoid, Pyramidoid and Spheroid. The volume of a Shapoid is defined by, for a Facoid:

$$\left\{\sum_{i \in I} v_i \overrightarrow{A_i} + \overrightarrow{P}\right\}, v_i \in [0.0, 1.0] \tag{1}$$

for a Pyramidoid:

$$\left\{\sum_{i \in I} v_i \overrightarrow{A_i} + \overrightarrow{P}\right\}, v_i \in [0.0, 1.0], \sum_{i \in I} v_i \leq 1.0 \tag{2}$$

and for a Spheroid:

$$\begin{array}{c} \left\{\sum_{i \in I} v_i \overrightarrow{A_i} + \overrightarrow{P}\right\}, \\ v_i \in [-0.5, 0.5], \quad \sum_{i \in I} v_i^2 \leq 0.25 \end{array} \tag{3}$$

1.1 Transformation

A translation of a Shapoid by $\overrightarrow{T}$ is obtained as follow:

$$\left(\overrightarrow{P}, \left\{\overrightarrow{A_i}\right\}_{i \in I}\right) \mapsto \left(\overrightarrow{P} + \overrightarrow{T}, \left\{\overrightarrow{A_i}\right\}_{i \in I}\right) \tag{4}$$

A scale of a Shapoid by $\overrightarrow{S}$ is obtained as follow:

$$\left(\overrightarrow{P}, \left\{\overrightarrow{A_i}\right\}_{i \in I}\right) \mapsto \left(\overrightarrow{P}, \left\{\overrightarrow{A_i'}\right\}_{i \in I}\right) \tag{5}$$

where

$$\overrightarrow{A_i'} = S_i \overrightarrow{A_i} \tag{6}$$

For Shapoid whose dimension D is equal to 2, a rotation by angle θ is obtained as follow:

$$\left(\overrightarrow{P}, \overrightarrow{A_0}, \overrightarrow{A_1}\right) \mapsto \left(\overrightarrow{P}, \overrightarrow{A_0'}, \overrightarrow{A_1'}\right) \tag{7}$$

where

$$\overrightarrow{A_i'} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \overrightarrow{A_i} \tag{8}$$

1.2 Shapoid's coordinate system

The Shapoid's coordinate system is the system having $\overrightarrow{P}$ as origin and $\overrightarrow{A_i}$ as axis. One can change from the Shapoid's coordinate system ($\overrightarrow{X^S}$) to the standard coordinate system ($\overrightarrow{X}$) as follow:

$$\overrightarrow{X} = \left[\left(\overrightarrow{A_0}\right)\left(\overrightarrow{A_1}\right)...\left(\overrightarrow{A_{D-1}}\right)\right]\overrightarrow{X^S} + \overrightarrow{P} \tag{9}$$

and reciprocally, from the standard coordinate system to the Shapoid's coordinate system:

$$\overrightarrow{X^S} = \left[\left(\overrightarrow{A_0}\right)\left(\overrightarrow{A_1}\right)...\left(\overrightarrow{A_{D-1}}\right)\right]^{-1}\left(\overrightarrow{X} - \overrightarrow{P}\right) \tag{10}$$

1.3 Insideness

$\overrightarrow{X}$ is inside the Shapoid S if, for a Facoid:

$$\forall i \in I, 0.0 \leq X_i^S \leq 1.0 \tag{11}$$

for a Pyramidoid:

$$\begin{cases} \forall i \in I, 0.0 \leq X_i^S \leq 1.0 \\ \sum_{i\in I} X_i^S \leq 1.0 \end{cases} \tag{12}$$

for a Spheroid:

$$\left|\left|\overrightarrow{X^S}\right|\right| \leq 0.5 \tag{13}$$

1.4 Bounding box

A bounding box of a Shapoid is a Facoid whose axis are colinear to axis of the standard coordinate system, and including the Shapoid in its volume. While the smallest possible bounding box can be easily obtained for Facoid and Pyramidoid, it's more complicate for Spheroid. Then we will consider for the Spheroid the bounding box of the equivalent Facoid $\left(\overrightarrow{P} - \sum_{i\in I}\left(0.5 * \overrightarrow{A_i}\right), \left\{\overrightarrow{A_i}\right\}_{i\in I}\right)$ which gives the smallest bounding box when axis of the Spheroid are colinear to axis of the standard coordinate system and a bounding box slightly too large when not colinear.
The bounding box is defined as follow, for a Facoid:

$$\left(\overrightarrow{P'}, \left\{\overrightarrow{A'_i}\right\}_{i\in I}\right) \tag{14}$$

where

$$\left\{\begin{array}{l} P'_i = P_i + \sum_{j\in I^-} A_{ji} \\ A'_{ij} = 0.0, i \neq j \\ A'_{ij} = \sum_{k\in I^+} A_{kj} - \sum_{k\in I^-} A_{kj}, i = j \end{array}\right. \tag{15}$$

and, I^+ and I^- are the subsets of I such as $\forall j \in I^+, A_{ij} \geq 0.0$ and $\forall j \in I^-, A_{ij} < 0.0$.

for a Pyramidoid:

$$\left(\overrightarrow{P'}, \left\{\overrightarrow{A'_i}\right\}_{i\in I}\right) \tag{16}$$

where

$$\left\{\begin{array}{l} P'_i = P_i + Min\left(Min_{j\in I}(A_{ji}), 0.0\right) \\ A'_{ij} = 0.0, i \neq j \\ A'_{ij} = Max_{k\in I}(A_{kj}) - Min_{k\in I}(A_{kj}), i = j \end{array}\right. \tag{17}$$

1.5 Depth and Center

Depth $\mathbf{D}_S(\overrightarrow{X})$ of position $\overrightarrow{X}$ a Shapoid S is a value ranging from 0.0 if $\overrightarrow{X}$ is on the surface of the Shapoid, to 1.0 if $\overrightarrow{X}$ is at the farthest location from the surface inside the Shapoid. Depth is by definition equal to 0.0 if $\overrightarrow{X}$ is outside the Shapoid. Depth is continuous and derivable on the volume of the Shapoid. It is defined by, for a Facoid:

$$\mathbf{D}_S(\overrightarrow{X}) = \prod_{i\in I} \left(1.0 - 4.0 * (0.5 - X_i^S)^2\right) \tag{18}$$

for a Pyramidoid:

$$\mathbf{D}_S(\overrightarrow{X}) = \prod_{i\in I} \left(1.0 - 4.0 * \left(0.5 - \frac{X_i^S}{1.0 - \sum_{j\in I-\{i\}} X_j^S}\right)^2\right) \tag{19}$$

and for a Spheroid:

$$\mathbf{D}_S(\overrightarrow{X}) = 1.0 - 2.0 * \left|\left|\overrightarrow{X^S}\right|\right| \tag{20}$$

The maximum depth is obtained at $\overrightarrow{C}$ such as, for a Facoid:

$$\forall i \in I, C_i^S = 0.5 \tag{21}$$

for a Pyramidoid:

$$\forall i \in I, C_i^S = \frac{1}{D+1} \tag{22}$$

for a Spheroid:

$$\forall i \in I, C_i^S = 0.0 \tag{23}$$

$\overrightarrow{C}$ is called the center of the Shapoid.

2 Interface

```
// ============ SHAPOID.H ================

#ifndef SHAPOID_H
#define SHAPOID_H

// ================= Include =================

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdbool.h>
#include "pberr.h"
#include "pbmath.h"
#include "gset.h"

// ================= Define ==================

#define SpheroidCreate(D) ShapoidCreate(D, ShapoidTypeSpheroid)
#define FacoidCreate(D) ShapoidCreate(D, ShapoidTypeFacoid)
#define PyramidoidCreate(D) ShapoidCreate(D, ShapoidTypePyramidoid)
#define ShapoidGetCoverage(A, B) ShapoidGetCoverageDelta(A, B, 0.1)

extern const char *ShapoidTypeString[3];

// ================= Polymorphism ==================

#define ShapoidGetBoundingBox(T) _Generic((T), \
  Shapoid*: ShapoidGetBoundingBoxThat, \
  GSet*: ShapoidGetBoundingBoxSet, \
  default: PBErrInvalidPolymorphism)(T)

#define ShapoidScale(T, C) _Generic((C), \
  VecFloat*: ShapoidScaleVector, \
  float: ShapoidScaleScalar, \
  default: PBErrInvalidPolymorphism)(T, C)

#define ShapoidGrow(T, C) _Generic((C), \
  VecFloat*: ShapoidGrowVector, \
  float: ShapoidGrowScalar, \
  default: PBErrInvalidPolymorphism)(T, C)

// ================= Data structure ===================
typedef enum ShapoidType {
  ShapoidTypeFacoid, ShapoidTypeSpheroid,
  ShapoidTypePyramidoid
} ShapoidType;
// Don’t forget to update ShapoidTypeString in shapoid.c when adding
// new type

typedef struct Shapoid {
  // Position of origin
  VecFloat *_pos;
  // Dimension
  int _dim;
  // Vectors defining axes
  VecFloat **_axis;
  // Type of Shapoid
  ShapoidType _type;
  // Linear sytem used to import coordinates
  SysLinEq *_sysLinEqImport;
} Shapoid;

// ================ Functions declaration ====================

// Create a Shapoid of dimension ’dim’ and type ’type’, default values:
// _pos = null vector
// _axis[d] = unit vector along dimension d
Shapoid* ShapoidCreate(int dim, ShapoidType type);

// Clone a Shapoid
Shapoid* ShapoidClone(Shapoid *that);

// Free memory used by a Shapoid
void ShapoidFree(Shapoid **that);

// Load the Shapoid from the stream
// If the Shapoid is already allocated, it is freed before loading
// Return true upon success else false
bool ShapoidLoad(Shapoid **that, FILE *stream);

// Save the Shapoid to the stream
// Return true upon success else false
bool ShapoidSave(Shapoid *that, FILE *stream);

// Print the Shapoid on ’stream’
void ShapoidPrintln(Shapoid *that, FILE *stream);

// Get the dimension of the Shapoid
#if BUILDMODE != 0
inline
#endif
int ShapoidGetDim(Shapoid *that);

// Get the type of the Shapoid
#if BUILDMODE != 0
inline
#endif
ShapoidType ShapoidGetType(Shapoid *that);

// Get the type of the Shapoid as a string
// Return a pointer to a constant string (not to be freed)
#if BUILDMODE != 0
inline
#endif
```

```
const char* ShapoidGetTypeAsString(Shapoid *that);

// Return a VecFloat equal to the position of the Shapoid
#if BUILDMODE != 0
inline
#endif
VecFloat* ShapoidGetPos(Shapoid *that);

// Return a VecFloat equal to the 'dim'-th axis of the Shapoid
#if BUILDMODE != 0
inline
#endif
VecFloat* ShapoidGetAxis(Shapoid *that, int dim);

// Set the position of the Shapoid to 'pos'
#if BUILDMODE != 0
inline
#endif
void ShapoidSetPos(Shapoid *that, VecFloat *pos);

// Set the 'dim'-th axis of the Shapoid to 'v'
#if BUILDMODE != 0
inline
#endif
void ShapoidSetAxis(Shapoid *that, int dim, VecFloat *v);

// Translate the Shapoid by 'v'
#if BUILDMODE != 0
inline
#endif
void ShapoidTranslate(Shapoid *that, VecFloat *v);

// Scale the Shapoid by 'v' (each axis is multiplied by v[iAxis])
#if BUILDMODE != 0
inline
#endif
void ShapoidScaleVector(Shapoid *that, VecFloat *v);

// Scale the Shapoid by 'c'
#if BUILDMODE != 0
inline
#endif
void ShapoidScaleScalar(Shapoid *that, float c);

// Scale the Shapoid by 'v' (each axis is multiplied by v[iAxis])
// and translate the Shapoid such as its center after scaling
// is at the same position than before scaling
#if BUILDMODE != 0
inline
#endif
void ShapoidGrowVector(Shapoid *that, VecFloat *v);

// Scale the Shapoid by 'c'
// and translate the Shapoid such as its center after scaling
// is at the same position than before scaling
#if BUILDMODE != 0
inline
#endif
void ShapoidGrowScalar(Shapoid *that, float c);

// Rotate the Shapoid of dimension 2 by 'theta' (in radians, CCW)
// relatively to its center
```

```
#if BUILDMODE != 0
inline
#endif
void ShapoidRotate2D(Shapoid *that, float theta);

// Convert the coordinates of 'pos' from standard coordinate system
// toward the Shapoid coordinates system
#if BUILDMODE != 0
inline
#endif
VecFloat* ShapoidImportCoord(Shapoid *that, VecFloat *pos);

// Convert the coordinates of 'pos' from the Shapoid coordinates system
// toward standard coordinate system
#if BUILDMODE != 0
inline
#endif
VecFloat* ShapoidExportCoord(Shapoid *that, VecFloat *pos);

// Return true if 'pos' (in stand coordinate system) is inside the
// Shapoid
// Else return false
bool ShapoidIsPosInside(Shapoid *that, VecFloat *pos);

// Get a bounding box of the Shapoid. The bounding box is aligned
// on the standard coordinate system (its axis are colinear with
// the axis of the standard coordinate system).
// The bounding box is returned as a Facoid, which position is
// at the minimum value along each axis.
Shapoid* ShapoidGetBoundingBoxThat(Shapoid *that);

// Get the bounding box of a set of Facoid. The bounding box is aligned
// on the standard coordinate system (its axis are colinear with
// the axis of the standard coordinate system).
// The bounding box is returned as a Facoid, which position is
// at the minimum value along each axis.
Shapoid* ShapoidGetBoundingBoxSet(GSet *set);

// Get the depth value in the Shapoid of 'pos' in standard coordinate
// system
// The depth is defined as follow: the point with depth equals 1.0 is
// the farthest point from the surface of the Shapoid (inside it),
// points with depth equals to 0.0 are point on the surface of the
// Shapoid. Depth is continuous and derivable over the volume of the
// Shapoid
// Return 0.0 if pos is outside the Shapoid
float ShapoidGetPosDepth(Shapoid *that, VecFloat *pos);

// Get the center of the shapoid in standard coordinate system
#if BUILDMODE != 0
inline
#endif
VecFloat* ShapoidGetCenter(Shapoid *that);

// Get the percentage of 'tho' included into 'that' (in [0.0, 1.0])
// 0.0 -> 'tho' is completely outside of 'that'
// 1.0 -> 'tho' is completely inside of 'that'
// 'that' and 'tho' must me of same dimensions
// delta is the step of the algorithm (in ]0.0, 1.0])
// small -> slow but precise
// big -> fast but rough
float ShapoidGetCoverageDelta(Shapoid *that, Shapoid *tho, float delta);
```

```
// Update the system of linear equation used to import coordinates
#if BUILDMODE != 0
inline
#endif
void ShapoidUpdateSysLinEqImport(Shapoid *that);

// Check if shapoid 'that' and 'tho' are equals
#if BUILDMODE != 0
inline
#endif
bool ShapoidIsEqual(Shapoid *that, Shapoid *tho);

// ================ Inliner ====================

#if BUILDMODE != 0
#include "shapoid-inline.c"
#endif


#endif
```

3 Code

3.1 shapoid.c

```
// ============ SHAPOID.C ================

// ================= Include =================

#include "shapoid.h"
#if BUILDMODE == 0
#include "shapoid-inline.c"
#endif

// ================= Define ==================

const char *ShapoidTypeString[3] = {
  (const char*)"Facoid", (const char*)"Spheroid",
  (const char*)"Pyramidoid"};

// ================ Functions implementation ====================

// Create a Shapoid of dimension 'dim' and type 'type', default values:
// _pos = null vector
// _axis[d] = unit vector along dimension d
Shapoid* ShapoidCreate(int dim, ShapoidType type) {
#if BUILDMODE == 0
  if (dim <= 0) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "Invalid dimension (%d>0)", dim);
    PBErrCatch(ShapoidErr);
  }
#endif
  // Declare a vector used for initialisation
  VecShort2D d = VecShortCreateStatic2D();
  // Declare a identity matrix used for initialisation
  VecSet(&d, 0, dim);
  VecSet(&d, 1, dim);
  MatFloat *mat = MatFloatCreate(&d);
  MatFloatSetIdentity(mat);
  // Allocate memory
  Shapoid *that = PBErrMalloc(ShapoidErr, sizeof(Shapoid));
  // Init pointers
  that->_pos = NULL;
  that->_axis = NULL;
  that->_sysLinEqImport = NULL;
  // Set the dimension and type
  that->_type = type;
  that->_dim = dim;
  // Allocate memory for position
  that->_pos = VecFloatCreate(dim);
  // Allocate memory for array of axis
  that->_axis = PBErrMalloc(ShapoidErr, sizeof(VecFloat*) * dim);
  for (int iAxis = dim; iAxis--;)
    that->_axis[iAxis] = NULL;
  // Allocate memory for each axis
  for (int iAxis = 0; iAxis < dim; ++iAxis) {
    // Allocate memory for position
    that->_axis[iAxis] = VecFloatCreate(dim);
    // Set value of the axis
    VecSet(that->_axis[iAxis], iAxis, 1.0);
  }
  // Create the linear system for coordinate importation
  that->_sysLinEqImport = SysLinEqCreate(mat, (VecFloat*)NULL);
  // Free memory
  MatFree(&mat);
  // Return the new Shapoid
  return that;
}

// Clone a Shapoid
Shapoid* ShapoidClone(Shapoid *that) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
#endif
  // Create a clone
  Shapoid *clone = ShapoidCreate(that->_dim, that->_type);
  // Set the position and axis of the clone
  ShapoidSetPos(clone, that->_pos);
  for (int iAxis = clone->_dim; iAxis--;)
    ShapoidSetAxis(clone, iAxis, that->_axis[iAxis]);
  // Clone the SysLinEq
  SysLinEqFree(&(clone->_sysLinEqImport));
  clone->_sysLinEqImport = SysLinEqClone(that->_sysLinEqImport);
  // Return the clone
  return clone;
}

// Free memory used by a Shapoid
void ShapoidFree(Shapoid **that) {
  // Check argument
  if (that == NULL || *that == NULL)
    return;
  // Free memory
  for (int iAxis = (*that)->_dim; iAxis--;)
    VecFree((*that)->_axis + iAxis);
  free((*that)->_axis);
  VecFree(&((*that)->_pos));
  SysLinEqFree(&((*that)->_sysLinEqImport));
  free(*that);
  *that = NULL;
}

// Load the Shapoid from the stream
// If the Shapoid is already allocated, it is freed before loading
// Return true upon success else false
bool ShapoidLoad(Shapoid **that, FILE *stream) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (stream == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'stream' is null");
    PBErrCatch(ShapoidErr);
  }
#endif
  // If 'that' is already allocated
  if (*that != NULL)
    // Free memory
    ShapoidFree(that);
  // Read the dimension and type
  int dim;
  int ret = fscanf(stream, "%d", &dim);
  // If we couldn't fscanf
  if (ret == EOF)
    return false;
  if (dim <= 0)
    return false;
  ShapoidType type;
  ret = fscanf(stream, "%u", &type);
  // If we coudln't fscanf
  if (ret == EOF)
    return false;
  // Allocate memory
  *that = ShapoidCreate(dim, type);
  // Read the values
  bool ok = VecFloatLoad(&((*that)->_pos), stream);
  if (ok == false)
    return false;
  for (int iAxis = 0; iAxis < dim; ++iAxis) {
    ok = VecFloatLoad((*that)->_axis + iAxis, stream);
    if (ok == false)
      return false;
  }
  // Update the SysLinEq
  ShapoidUpdateSysLinEqImport(*that);
  // Return success code
  return true;
}

// Save the Shapoid to the stream
// Return true upon success else false
```

```
bool ShapoidSave(Shapoid *that, FILE *stream) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (stream == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'stream' is null");
    PBErrCatch(ShapoidErr);
  }
#endif
  // Save the dimension and type
  int ret = fprintf(stream, "%d %u\n", that->_dim, that->_type);
  // If we coudln't fprintf
  if (ret < 0)
    return false;
  // Save the position and axis
  bool ok = VecFloatSave(that->_pos, stream);
  if (ok == false)
    return false;
  for (int iAxis = 0; iAxis < that->_dim; ++iAxis) {
    ok = VecFloatSave(that->_axis[iAxis], stream);
    if (ok == false)
      return false;
  }
  // Return success code
  return true;
}

// Print the Shapoid on 'stream'
void ShapoidPrintln(Shapoid *that, FILE *stream) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (stream == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'stream' is null");
    PBErrCatch(ShapoidErr);
  }
#endif
  // Print the Shapoid
  fprintf(stream, "Type: %s\n", ShapoidTypeString[that->_type]);
  fprintf(stream, "Dim: %d\n", that->_dim);
  fprintf(stream, "Pos: ");
  VecPrint(that->_pos, stream);
  fprintf(stream, "\n");
  for (int iAxis = 0; iAxis < that->_dim; ++iAxis) {
    fprintf(stream, "Axis(%d): ", iAxis);
    VecPrint(that->_axis[iAxis], stream);
    fprintf(stream, "\n");
  }
}

// Return true if 'pos' (in stand coordinate system) is inside the
// Shapoid
// Else return false
bool ShapoidIsPosInside(Shapoid *that, VecFloat *pos) {
```

```
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (pos == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'pos' is null");
    PBErrCatch(ShapoidErr);
  }
  if (VecDim(pos) != that->_dim) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "'pos' 's dimension is invalid (%d==%d)",
      that->_dim, VecDim(pos));
    PBErrCatch(ShapoidErr);
  }
  if (that->_type != ShapoidTypeFacoid &&
    that->_type != ShapoidTypeSpheroid &&
    that->_type != ShapoidTypePyramidoid) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "No implementation for 'that' 's type");
    PBErrCatch(ShapoidErr);
  }
#endif
  // Get the coordinates of pos in the Shapoid coordinate system
  VecFloat *coord = ShapoidImportCoord(that, pos);
  // Declare a variable to memorize the result
  bool ret = false;
  // If the Shapoid is a Facoid
  if (that->_type == ShapoidTypeFacoid) {
    // pos is in the Shapoid if all the coord in Shapoid coord
    // system are in [0.0, 1.0]
    ret = true;
    for (int dim = that->_dim; dim-- && ret == true;) {
      float v = VecGet(coord, dim);
      if (v < 0.0 || v > 1.0)
        ret = false;
    }
  // Else, if the Shapoid is a Pyramidoid
  } else if (that->_type == ShapoidTypePyramidoid) {
    // pos is in the Shapoid if all the coord in Shapoid coord
    // system are in [0.0, 1.0] and their sum is in [0.0, 1.0]
    ret = true;
    float sum = 0.0;
    for (int dim = that->_dim; dim-- && ret == true;) {
      float v = VecGet(coord, dim);
      sum += v;
      if (v < 0.0 || v > 1.0)
        ret = false;
    }
    if (ret == true && sum > 1.0)
      ret = false;
  // Else, if the Shapoid is a Spheroid
  } else if (that->_type == ShapoidTypeSpheroid) {
    // pos is in the Shapoid if its norm is in [0.0, 0.5]
    float norm = VecNorm(coord);
    if (norm <= 0.5)
      ret = true;
  }
  // Free memory
  VecFloatFree(&coord);
  // Return the result
  return ret;
}

// Get a bounding box of the Shapoid. The bounding box is aligned
// on the standard coordinate system (its axis are colinear with
// the axis of the standard coordinate system).
// The bounding box is returned as a Facoid, which position is
// at the minimum value along each axis.
Shapoid* ShapoidGetBoundingBoxThat(Shapoid *that) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (that->_type != ShapoidTypeFacoid &&
    that->_type != ShapoidTypeSpheroid &&
    that->_type != ShapoidTypePyramidoid) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "No implementation for 'that' 's type");
    PBErrCatch(ShapoidErr);
  }
#endif
  // Declare a variable to memorize the result
  Shapoid *res = FacoidCreate(ShapoidGetDim(that));
  // If the Shapoid is a Facoid
  if (that->_type == ShapoidTypeFacoid) {
    // For each axis
    for (int dim = that->_dim; dim--;) {
      // Declare a variable to memorize the bound of the interval on
      // this axis
      float bound[2];
      bound[0] = bound[1] = VecGet(that->_pos, dim);
      // For each parameter
      for (int param = that->_dim; param--;) {
        // Get the value of the axis influencing the current dimension
        float v = VecGet(that->_axis[param], dim);
        // If the value is negative, update the minimum bound
        if (v < 0.0)
          bound[0] += v;
        // Else, if the value is negative, update the minimum bound
        else
          bound[1] += v;
      }
      // Memorize the result
      VecSet(res->_pos, dim, bound[0]);
      VecSet(res->_axis[dim], dim, bound[1] - bound[0]);
    }
  // Else, if the Shapoid is a Pyramidoid
  } else  if (that->_type == ShapoidTypePyramidoid) {
    // For each axis
    for (int dim = that->_dim; dim--;) {
      // Declare a variable to memorize the bound of the interval on
      // this axis
      float bound[2];
      bound[0] = bound[1] = VecGet(that->_pos, dim);
      // For each parameter
      for (int param = that->_dim; param--;) {
        // Get the value of the axis influencing the current dimension
        float v = VecGet(that->_axis[param], dim);
        // Search the min and max values
        if (v < bound[0])
          bound[0] = v;
        if (v > bound[1])
          bound[1] = v;
      }
      // Memorize the result
      VecSet(res->_pos, dim, bound[0]);
      VecSet(res->_axis[dim], dim, bound[1] - bound[0]);
    }
  // Else, if the Shapoid is a Spheroid
  } else  if (that->_type == ShapoidTypeSpheroid) {
    // In case of a Spheroid, things get complicate
    // We'll approximate the bounding box of the Spheroid
    // with the one of the same Spheroid viewed as a Facoid
    // and simply take care that the _pos is at the center of the
    // Spheroid
    // For each axis
    for (int dim = that->_dim; dim--;) {
      // Declare a variable to memorize the bound of the interval on
      // this axis
      float bound[2];
      bound[0] = VecGet(that->_pos, dim);
      // Correct position
      // For each parameter
      for (int param = that->_dim; param--;) {
        // Get the value of the axis influencing the current dimension
        float v = VecGet(that->_axis[param], dim);
        // Correct the pos
        bound[0] -= 0.5 * v;
      }
      bound[1] = bound[0];
      // For each parameter
      for (int param = that->_dim; param--;) {
        // Get the value of the axis influencing the current dimension
        float v = VecGet(that->_axis[param], dim);
        // If the value is negative, update the minimum bound
        if (v < 0.0)
          bound[0] += v;
        // Else, if the value is negative, update the minimum bound
        else
          bound[1] += v;
      }
      // Memorize the result
      VecSet(res->_pos, dim, bound[0]);
      VecSet(res->_axis[dim], dim, bound[1] - bound[0]);
    }
  }
  // Return the result
  return res;
}

// Get the bounding box of a set of Facoid. The bounding box is aligned
// on the standard coordinate system (its axis are colinear with
// the axis of the standard coordinate system).
// The bounding box is returned as a Facoid, which position is
// at the minimum value along each axis.
Shapoid* ShapoidGetBoundingBoxSet(GSet *set) {
#if BUILDMODE == 0
  if (set == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'set' is null");
    PBErrCatch(ShapoidErr);
  }
  GSetElem *elemCheck = set->_head;
  int dim = ((Shapoid*)(elemCheck->_data))->_dim;
  while (elemCheck != NULL) {
    if (((Shapoid*)(elemCheck->_data))->_dim != dim) {
      ShapoidErr->_type = PBErrTypeInvalidArg;
      sprintf(ShapoidErr->_msg,
        "'set' contains Shapoids of various dimensions");
      PBErrCatch(ShapoidErr);
    }
    elemCheck = elemCheck->_next;
  }
#endif
  // Declare a variable for the result
  Shapoid *res = NULL;
  // Declare a pointer to the elements of the set
  GSetElem *elem = set->_head;
  // Loop on element of the set
  while (elem != NULL) {
    // Declare a pointer to the Facoid
    Shapoid *shapoid = (Shapoid*)(elem->_data);
    // If it's the first Facoid in the set
    if (res == NULL) {
      // Get the bounding box of this shapoid
      res = ShapoidGetBoundingBox(shapoid);
    // Else, this is not the first Shapoid in the set
    } else {
      // Get the bounding box of this shapoid
      Shapoid *bound = ShapoidGetBoundingBox(shapoid);
      // For each dimension
      for (int iDim = res->_dim; iDim--;) {
        // Update the bounding box
        if (VecGet(bound->_pos, iDim) < VecGet(res->_pos, iDim)) {
          VecSet(res->_axis[iDim], iDim,
            VecGet(res->_axis[iDim], iDim) +
            VecGet(res->_pos, iDim) -
            VecGet(bound->_pos, iDim));
          VecSet(res->_pos, iDim, VecGet(bound->_pos, iDim));
        }
        if (VecGet(bound->_pos, iDim) +
          VecGet(bound->_axis[iDim], iDim) >
          VecGet(res->_pos, iDim) +
          VecGet(res->_axis[iDim], iDim))
          VecSet(res->_axis[iDim], iDim,
          VecGet(bound->_pos, iDim) +
          VecGet(bound->_axis[iDim], iDim) -
          VecGet(res->_pos, iDim));
      }
      // Free memory used by the bounding box
      ShapoidFree(&bound);
    }
    // Move to the next element
    elem = elem->_next;
  }
  // Return the result
  return res;
}

// Get the depth value in the Shapoid of 'pos'
// The depth is defined as follow: the point with depth equals 1.0 is
// the farthest point from the surface of the Shapoid (inside it),
// points with depth equals to 0.0 are point on the surface of the
// Shapoid. Depth is continuous and derivable over the volume of the
// Shapoid
float ShapoidGetPosDepth(Shapoid *that, VecFloat *pos) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (pos == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'pos' is null");
    PBErrCatch(ShapoidErr);
  }
  if (VecDim(pos) != that->_dim) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "'pos' 's dimension is invalid (%d==%d)",
      that->_dim, VecDim(pos));
    PBErrCatch(ShapoidErr);
  }
  if (that->_type != ShapoidTypeFacoid &&
    that->_type != ShapoidTypeSpheroid &&
    that->_type != ShapoidTypePyramidoid) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "No implementation for 'that' 's type");
    PBErrCatch(ShapoidErr);
  }
#endif
  // Get the coordinates of pos in the Shapoid coordinate system
  VecFloat *coord = ShapoidImportCoord(that, pos);
  // Declare a variable to memorize the result
  float ret = 0.0;
  // If the Shapoid is a Facoid
  if (that->_type == ShapoidTypeFacoid) {
    ret = 1.0;
    for (int dim = that->_dim; dim-- && ret > PBMATH_EPSILON;) {
      float v = VecGet(coord, dim);
      if (v < 0.0 || VecGet(coord, dim) > 1.0)
        ret = 0.0;
      else
        ret *= 1.0 - pow(0.5 - v, 2.0) * 4.0;
    }
  // Else, if the Shapoid is a Pyramidoid
  } else if (that->_type == ShapoidTypePyramidoid) {
    ret = 1.0;
    float sum = 0.0;
    bool flag = true;
    for (int dim = that->_dim; dim-- && ret > PBMATH_EPSILON;) {
      float v = VecGet(coord, dim);
      sum += v;
      if (v < 0.0 || v > 1.0)
        flag = false;
    }
    if (flag == true && sum > 1.0)
      flag = false;
    if (flag == false)
      ret = 0.0;
    else {
      ret = 1.0;
      for (int dim = ShapoidGetDim(that); dim--;) {
        float z = 0.0;
        for (int d = ShapoidGetDim(that); d--;)
```

```
          if (d != dim)
            z += VecGet(coord, d);
        ret *=
          (1.0 - 4.0 * pow(0.5 - VecGet(coord, dim) / (1.0 - z), 2.0));
      }
    }
  // Else, if the Shapoid is a Spheroid
  } else if (that->_type == ShapoidTypeSpheroid) {
    float norm = VecNorm(coord);
    if (norm <= 0.5)
      ret = 1.0 - norm * 2.0;
  }
  // Free memory
  VecFloatFree(&coord);
  // Return the result
  return ret;
}

// Get the percentage of 'tho' included 'that' (in [0.0, 1.0])
// 0.0 -> 'tho' is completely outside of 'that'
// 1.0 -> 'tho' is completely inside of 'that'
// 'that' and 'tho' must me of same dimensions
// delta is the step of the algorithm (in ]0.0, 1.0])
// small -> slow but precise
// big -> fast but rough
float ShapoidGetCoverageDelta(Shapoid *that, Shapoid *tho, float delta) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (tho == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'tho' is null");
    PBErrCatch(ShapoidErr);
  }
  if (ShapoidGetDim(that) != ShapoidGetDim(tho)) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg,
      "Shapoids dimensions are different (%d==%d)",
      ShapoidGetDim(that), ShapoidGetDim(tho));
    PBErrCatch(ShapoidErr);
  }
  if (that->_type != ShapoidTypeFacoid &&
    that->_type != ShapoidTypeSpheroid &&
    that->_type != ShapoidTypePyramidoid) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "No implementation for 'that' 's type");
    PBErrCatch(ShapoidErr);
  }
#endif
  // Declare variables to compute the result
  float ratio = 0.0;
  float sum = 0.0;
  // Declare variables for the relative and absolute position in 'tho'
  VecFloat *pRel = VecFloatCreate(ShapoidGetDim(that));
  VecFloat *pStd = NULL;
  // Declare a variable to memorize the last index in dimension
  int lastI = VecDim(pRel) - 1;
  // Declare a variable to memorize the max value of coordinates
  float max = 1.0;
  // If 'tho' is a spheroid, correct the start coordinates and range
  if (tho->_type == ShapoidTypeSpheroid) {
    max = 0.5;
    for (int iDim = ShapoidGetDim(that); iDim--;)
      VecSet(pRel, iDim, -0.5);
  }
  // Loop on relative coordinates
  while (VecGet(pRel, lastI) <= max + PBMATH_EPSILON) {
    // Get the coordinates in standard system
    pStd = ShapoidExportCoord(tho, pRel);
    // If this position is inside 'tho'
    if (ShapoidIsPosInside(tho, pStd) == true) {
      // If this position is inside 'that'
      if (ShapoidIsPosInside(that, pStd) == true)
        // Increment the ratio
        ratio += 1.0;
      sum += 1.0;
    }
    // Free memory
    VecFree(&pStd);
    // Step the relative coordinates
    int iDim = 0;
    while (iDim >= 0) {
      VecSet(pRel, iDim, VecGet(pRel, iDim) + delta);
      if (iDim != lastI &&
        VecGet(pRel, iDim) > max + PBMATH_EPSILON) {
        VecSet(pRel, iDim, max - 1.0);
        ++iDim;
      } else {
        iDim = -1;
      }
    }
  }
  // Finish the computation of the ratio
  ratio /= sum;
  // Free memory
  VecFree(&pRel);
  // Return the result
  return ratio;
}
```

3.2 shapoid-inline.c

```
// ============ SHAPOID-INLINE.C ================

// ================ Functions implementation ====================

// Get the dimension of the Shapoid
#if BUILDMODE != 0
inline
#endif
int ShapoidGetDim(Shapoid *that) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
#endif
```

```
  // Return the dimension
  return that->_dim;
}

// Get the dimension of the Shapoid
#if BUILDMODE != 0
inline
#endif
ShapoidType ShapoidGetType(Shapoid *that) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
#endif
  // Return the type
  return that->_type;
}

// Get the type of the Shapoid as a string
// Return a pointer to a constant string (not to be freed)
#if BUILDMODE != 0
inline
#endif
const char* ShapoidGetTypeAsString(Shapoid *that) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
#endif
  // Return the type
  return ShapoidTypeString[that->_type];
}

// Return a VecFloat equal to the position of the Shapoid
#if BUILDMODE != 0
inline
#endif
VecFloat* ShapoidGetPos(Shapoid *that) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
#endif
  // Return a clone of the position
  return VecClone(that->_pos);
}

// Return a VecFloat equal to the 'dim'-th axis of the Shapoid
#if BUILDMODE != 0
inline
#endif
VecFloat* ShapoidGetAxis(Shapoid *that, int dim) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (dim < 0 || dim >= that->_dim) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "Axis' index is invalid (0<=%d<%d)",
      dim, that->_dim);
    PBErrCatch(ShapoidErr);
  }
#endif
  // Return a clone of the axis
  return VecClone(that->_axis[dim]);
}

// Set the position of the Shapoid to 'pos'
#if BUILDMODE != 0
inline
#endif
void ShapoidSetPos(Shapoid *that, VecFloat *pos) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (pos == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'pos' is null");
    PBErrCatch(ShapoidErr);
  }
  if (VecDim(pos) != that->_dim) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "'pos' 's dimension is invalid (%d==%d)",
      VecDim(pos), that->_dim);
    PBErrCatch(ShapoidErr);
  }
#endif
  // Set the position
  VecCopy(that->_pos, pos);
}

// Set the 'dim'-th axis of the Shapoid to 'v'
#if BUILDMODE != 0
inline
#endif
void ShapoidSetAxis(Shapoid *that, int dim, VecFloat *v) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (v == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'v' is null");
    PBErrCatch(ShapoidErr);
  }
  if (dim < 0 || dim >= that->_dim) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "Axis' index is invalid (0<=%d<%d)",
      dim, that->_dim);
    PBErrCatch(ShapoidErr);
  }
  if (VecDim(v) != that->_dim) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "'v' 's dimension is invalid (%d==%d)",
      dim, VecDim(v));
    PBErrCatch(ShapoidErr);
  }
#endif
  // Set the axis
  VecCopy(that->_axis[dim], v);
  // Update the SysLinEq
  ShapoidUpdateSysLinEqImport(that);
}

// Translate the Shapoid by 'v'
#if BUILDMODE != 0
inline
#endif
void ShapoidTranslate(Shapoid *that, VecFloat *v) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (v == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'v' is null");
    PBErrCatch(ShapoidErr);
  }
  if (VecDim(v) != that->_dim) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "'v' 's dimension is invalid (%d==%d)",
      that->_dim, VecDim(v));
    PBErrCatch(ShapoidErr);
  }
#endif
  // Translate the position
  VecOp(that->_pos, 1.0, v, 1.0);
}

// Scale the Shapoid by 'v' (each axis is multiplied by v[iAxis])
#if BUILDMODE != 0
inline
#endif
void ShapoidScaleVector(Shapoid *that, VecFloat *v) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (v == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'v' is null");
    PBErrCatch(ShapoidErr);
  }
  if (VecDim(v) != that->_dim) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "'v' 's dimension is invalid (%d==%d)",
      that->_dim, VecDim(v));
    PBErrCatch(ShapoidErr);
  }
```

```
#endif
  // Scale each axis
  for (int iAxis = that->_dim; iAxis--;)
    VecScale(that->_axis[iAxis], VecGet(v, iAxis));
  // Update the SysLinEq
  ShapoidUpdateSysLinEqImport(that);
}

// Scale the Shapoid by 'c'
#if BUILDMODE != 0
inline
#endif
void ShapoidScaleScalar(Shapoid *that, float c) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
#endif
  // Scale each axis
  for (int iAxis = that->_dim; iAxis--;)
    VecScale(that->_axis[iAxis], c);
  // Update the SysLinEq
  ShapoidUpdateSysLinEqImport(that);
}

// Scale the Shapoid by 'v' (each axis is multiplied by v[iAxis])
// and translate the Shapoid such as its center after scaling
// is at the same position than before scaling
#if BUILDMODE != 0
inline
#endif
void ShapoidGrowVector(Shapoid *that, VecFloat *v) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (v == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'v' is null");
    PBErrCatch(ShapoidErr);
  }
  if (VecDim(v) != that->_dim) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "'v' 's dimension is invalid (%d==%d)",
      that->_dim, VecDim(v));
    PBErrCatch(ShapoidErr);
  }
  if (that->_type != ShapoidTypeFacoid &&
    that->_type != ShapoidTypeSpheroid &&
    that->_type != ShapoidTypePyramidoid) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "No implementation for 'that' 's type");
    PBErrCatch(ShapoidErr);
  }
#endif
  // If the shapoid is a spheroid
  if (that->_type == ShapoidTypeSpheroid) {
    // Scale
    ShapoidScale(that, v);
  // Else, the shapoid is not a spheroid
  } else {
    // Memorize the center
    VecFloat *centerA = ShapoidGetCenter(that);
    // Scale
    ShapoidScale(that, v);
    // Reposition to keep center at the same position
    VecFloat *centerB = ShapoidGetCenter(that);
    VecOp(centerA, 1.0, centerB, -1.0);
    VecOp(that->_pos, 1.0, centerA, 1.0);
    VecFree(&centerA);
    VecFree(&centerB);
  }
  // Update the SysLinEq
  ShapoidUpdateSysLinEqImport(that);
}

// Scale the Shapoid by 'c'
// and translate the Shapoid such as its center after scaling
// is at the same position than before scaling
#if BUILDMODE != 0
inline
#endif
void ShapoidGrowScalar(Shapoid *that, float c) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (that->_type != ShapoidTypeFacoid &&
    that->_type != ShapoidTypeSpheroid &&
    that->_type != ShapoidTypePyramidoid) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "No implementation for 'that' 's type");
    PBErrCatch(ShapoidErr);
  }
#endif
  // If the shapoid is a spheroid
  if (that->_type == ShapoidTypeSpheroid) {
    // Scale
    ShapoidScale(that, c);
  // Else, the shapoid is not a spheroid
  } else {
    // Memorize the center
    VecFloat *centerA = ShapoidGetCenter(that);
    // Scale
    ShapoidScale(that, c);
    // Reposition to keep center at the same position
    VecFloat *centerB = ShapoidGetCenter(that);
    VecOp(centerA, 1.0, centerB, -1.0);
    VecOp(that->_pos, 1.0, centerA, 1.0);
    VecFree(&centerA);
    VecFree(&centerB);
  }
  // Update the SysLinEq
  ShapoidUpdateSysLinEqImport(that);
}


// Rotate the Shapoid of dimension 2 by 'theta' (in radians, CCW)
// relatively to its center
#if BUILDMODE != 0
inline
#endif
void ShapoidRotate2D(Shapoid *that, float theta) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (that->_dim != 2) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "'that' 's dimension is invalid (%d==2)",
      that->_dim);
    PBErrCatch(ShapoidErr);
  }
  if (that->_type != ShapoidTypeFacoid &&
    that->_type != ShapoidTypeSpheroid &&
    that->_type != ShapoidTypePyramidoid) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "No implementation for 'that' 's type");
    PBErrCatch(ShapoidErr);
  }
#endif
  // If it's a spheroid
  if (that->_type == ShapoidTypeSpheroid) {
  // Rotate each axis
  for (int iAxis = that->_dim; iAxis--;)
    VecRot(that->_axis[iAxis], theta);
  // Else, it's not a spheroid
  } else {
    VecFloat *center = ShapoidGetCenter(that);
    // Rotate each axis
    for (int iAxis = that->_dim; iAxis--;)
      VecRot(that->_axis[iAxis], theta);
    // Reposition the origin
    VecFloat *v = VecGetOp(that->_pos, 1.0, center, -1.0);
    VecRot(v, theta);
    VecOp(v, 1.0, center, 1.0);
    VecCopy(that->_pos, v);
    VecFree(&center);
    VecFree(&v);
  }
  // Update the SysLinEq
  ShapoidUpdateSysLinEqImport(that);
}

// Convert the coordinates of 'pos' from standard coordinate system
// toward the Shapoid coordinates system
#if BUILDMODE != 0
inline
#endif
VecFloat* ShapoidImportCoord(Shapoid *that, VecFloat *pos) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (pos == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'pos' is null");
    PBErrCatch(ShapoidErr);
  }
  if (VecDim(pos) != that->_dim) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "'pos' 's dimension is invalid (%d==%d)",
      that->_dim, VecDim(pos));
    PBErrCatch(ShapoidErr);
  }
#endif
  // Update the system solver for the requested position
  VecFloat *v = VecGetOp(pos, 1.0, that->_pos, -1.0);
  SysLinEqSetV(that->_sysLinEqImport, v);
  // Solve the system
  VecFloat *res = SysLinEqSolve(that->_sysLinEqImport);
  // Free memory
  VecFree(&v);
  // return the result
  return res;
}

// Convert the coordinates of 'pos' from the Shapoid coordinates system
// toward standard coordinate system
#if BUILDMODE != 0
inline
#endif
VecFloat* ShapoidExportCoord(Shapoid *that, VecFloat *pos) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (pos == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'pos' is null");
    PBErrCatch(ShapoidErr);
  }
  if (VecDim(pos) != that->_dim) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "'pos' 's dimension is invalid (%d==%d)",
      that->_dim, VecDim(pos));
    PBErrCatch(ShapoidErr);
  }
#endif
  // Allocate memory for the result
  VecFloat *res = VecClone(that->_pos);
  for (int dim = that->_dim; dim--;)
    VecOp(res, 1.0, that->_axis[dim], VecGet(pos, dim));
  // Return the result
  return res;
}

// Get the center of the shapoid in standard coordinate system
#if BUILDMODE != 0
inline
#endif
VecFloat* ShapoidGetCenter(Shapoid *that) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (that->_type != ShapoidTypeFacoid &&
    that->_type != ShapoidTypeSpheroid &&
    that->_type != ShapoidTypePyramidoid) {
    ShapoidErr->_type = PBErrTypeInvalidArg;
    sprintf(ShapoidErr->_msg, "No implementation for 'that' 's type");
    PBErrCatch(ShapoidErr);
  }
#endif
  // Declare a variable to memorize the result in Shapoid
  // coordinate system
  VecFloat *coord = VecFloatCreate(ShapoidGetDim(that));
  // If we could allocate memory
  if (coord != NULL) {
    // For each dimension
    for (int dim = ShapoidGetDim(that); dim--;) {
      if (ShapoidGetType(that) == ShapoidTypeFacoid)
        VecSet(coord, dim, 0.5);
      else if (ShapoidGetType(that) == ShapoidTypePyramidoid)
        VecSet(coord, dim, 1.0 / (1.0 + ShapoidGetDim(that)));
      else if (ShapoidGetType(that) == ShapoidTypeSpheroid)
        VecSet(coord, dim, 0.0);
    }
  }
  // Convert the coordinates in standard coordinate system
  VecFloat *res = ShapoidExportCoord(that, coord);
  // Free memory
  VecFree(&coord);
  // Return the result
  return res;
}

// Check if shapoid 'that' and 'tho' are equals
#if BUILDMODE != 0
inline
#endif
bool ShapoidIsEqual(Shapoid *that, Shapoid *tho) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
  if (tho == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'tho' is null");
    PBErrCatch(ShapoidErr);
  }
#endif
  // Check the dimension, type and position
  if (that->_dim != tho->_dim ||
    that->_type != tho->_type ||
    VecIsEqual(that->_pos, tho->_pos) == false)
    return false;
  // Check the axis
  for (int i = that->_dim; i--;)
    if (VecIsEqual(that->_axis[i], tho->_axis[i]) == false)
      return false;
  // Return the success code
  return true;
}
```

```
// Update the system of linear equation used to import coordinates
#if BUILDMODE != 0
inline
#endif
void ShapoidUpdateSysLinEqImport(Shapoid *that) {
#if BUILDMODE == 0
  if (that == NULL) {
    ShapoidErr->_type = PBErrTypeNullPointer;
    sprintf(ShapoidErr->_msg, "'that' is null");
    PBErrCatch(ShapoidErr);
  }
#endif
  VecShort2D dim = VecShortCreateStatic2D();
  // Set a pointer to the matrix in the SysLinEq
  MatFloat *mat = MatClone(that->_sysLinEqImport->_M);
  // Set the values of the matrix
  for (VecSet(&dim, 0, 0); VecGet(&dim, 0) < that->_dim;
    VecSet(&dim, 0, VecGet(&dim, 0) + 1)) {
    for (VecSet(&dim, 1, 0); VecGet(&dim, 1) < that->_dim;
      VecSet(&dim, 1, VecGet(&dim, 1) + 1)) {
      MatSet(mat, &dim, VecGet(that->_axis[VecGet(&dim, 0)],
        VecGet(&dim, 1)));
    }
  }
  // Update the SysLinEq
  SysLinEqSetM(that->_sysLinEqImport, mat);
  // Free memory
  MatFree(&mat);
}
```

4 Makefile

```
#directory
PBERRDIR=../PBErr
PBMATHDIR=../PBMath
GSETDIR=../GSet

# Build mode
# 0: development (max safety, no optimisation)
# 1: release (min safety, optimisation)
# 2: fast and furious (no safety, optimisation)
BUILDMODE=0

include $(PBERRDIR)/Makefile.inc

INCPATH=-I./ -I$(PBERRDIR)/ -I$(PBMATHDIR)/ -I$(GSETDIR)/
BUILDOPTIONS=$(BUILDPARAM) $(INCPATH)

# compiler
COMPILER=gcc

#rules
all : main

main: main.o pberr.o shapoid.o Makefile pbmath.o gset.o
$(COMPILER) main.o pberr.o shapoid.o  pbmath.o gset.o $(LINKOPTIONS) -o main
```

```
main.o : main.c $(PBERRDIR)/pberr.h shapoid.h shapoid-inline.c Makefile
$(COMPILER) $(BUILDOPTIONS) -c main.c

shapoid.o : shapoid.c shapoid.h shapoid-inline.c $(PBMATHDIR)/pbmath.h $(GSETDIR)/gset.h $(PBERRDIR)/pberr.h Makefil
$(COMPILER) $(BUILDOPTIONS) -c shapoid.c

pberr.o : $(PBERRDIR)/pberr.c $(PBERRDIR)/pberr.h Makefile
$(COMPILER) $(BUILDOPTIONS) -c $(PBERRDIR)/pberr.c

pbmath.o : $(PBMATHDIR)/pbmath.c $(PBMATHDIR)/pbmath-inline.c $(PBMATHDIR)/pbmath.h Makefile $(PBERRDIR)/pberr.h
$(COMPILER) $(BUILDOPTIONS) -c $(PBMATHDIR)/pbmath.c

gset.o : $(GSETDIR)/gset.c $(GSETDIR)/gset-inline.c $(GSETDIR)/gset.h Makefile $(PBERRDIR)/pberr.h
$(COMPILER) $(BUILDOPTIONS) -c $(GSETDIR)/gset.c

clean :
rm -rf *.o main

valgrind :
valgrind -v --track-origins=yes --leak-check=full --gen-suppressions=yes --show-leak-kinds=all ./main

unitTest :
main > unitTest.txt; diff unitTest.txt unitTestRef.txt
```

5 Unit tests

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>
#include "pberr.h"
#include "shapoid.h"

#define RANDOMSEED 0

void UnitTestCreateCloneIsEqualFree() {
  int dim = 3;
  Shapoid *facoid = ShapoidCreate(dim, ShapoidTypeFacoid);
  if (facoid == NULL || facoid->_dim != dim ||
    facoid->_type != ShapoidTypeFacoid || facoid->_pos == NULL ||
    VecDim(facoid->_pos) != dim || facoid->_sysLinEqImport == NULL ||
    facoid->_axis == NULL) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidCreate failed");
    PBErrCatch(ShapoidErr);
  }
  for (int iDim = dim; iDim--;) {
    if (ISEQUALF(VecGet(facoid->_pos, iDim), 0.0) == false ||
      facoid->_axis[iDim] == NULL ||
      VecDim(facoid->_axis[iDim]) != dim) {
      ShapoidErr->_type = PBErrTypeUnitTestFailed;
      sprintf(ShapoidErr->_msg, "ShapoidCreate failed");
      PBErrCatch(ShapoidErr);
    }
    for (int jDim = dim; jDim--;) {
      if ((iDim == jDim &&
        ISEQUALF(VecGet(facoid->_axis[iDim], jDim), 1.0) == false) ||
        (iDim != jDim &&
        ISEQUALF(VecGet(facoid->_axis[iDim], jDim), 0.0) == false)) {
        ShapoidErr->_type = PBErrTypeUnitTestFailed;
        sprintf(ShapoidErr->_msg, "ShapoidCreate failed");
        PBErrCatch(ShapoidErr);
      }
    }
    if (ISEQUALF(VecGet(facoid->_sysLinEqImport->_V, iDim),
      0.0) == false) {
      ShapoidErr->_type = PBErrTypeUnitTestFailed;
      sprintf(ShapoidErr->_msg, "ShapoidCreate failed");
      PBErrCatch(ShapoidErr);
    }
  }
  VecShort2D u = VecShortCreateStatic2D();
  VecSet(&u, 0, dim); VecSet(&u, 1, dim);
  VecShort2D v = VecShortCreateStatic2D();
  do {
    if ((VecGet(&v, 0) == VecGet(&v, 1) &&
      ISEQUALF(MatGet(facoid->_sysLinEqImport->_M, &v), 1.0) == false) ||
      (VecGet(&v, 0) != VecGet(&v, 1) &&
      ISEQUALF(MatGet(facoid->_sysLinEqImport->_M, &v), 0.0) == false) ||
      (VecGet(&v, 0) == VecGet(&v, 1) &&
      ISEQUALF(MatGet(facoid->_sysLinEqImport->_Minv, &v),
        1.0) == false) ||
      (VecGet(&v, 0) != VecGet(&v, 1) &&
      ISEQUALF(MatGet(facoid->_sysLinEqImport->_Minv, &v),
        0.0) == false)) {
      ShapoidErr->_type = PBErrTypeUnitTestFailed;
      sprintf(ShapoidErr->_msg, "ShapoidCreate failed");
      PBErrCatch(ShapoidErr);
    }
  } while (VecStep(&v, &u));
  Shapoid *clone = ShapoidClone(facoid);
  if (facoid->_dim != clone->_dim ||
    facoid->_type != clone->_type ||
    VecIsEqual(facoid->_pos, clone->_pos) == false ||
    MatIsEqual(facoid->_sysLinEqImport->_M,
      clone->_sysLinEqImport->_M) == false ||
    MatIsEqual(facoid->_sysLinEqImport->_Minv,
      clone->_sysLinEqImport->_Minv) == false ||
    VecIsEqual(facoid->_sysLinEqImport->_V,
      clone->_sysLinEqImport->_V) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidClone failed");
    PBErrCatch(ShapoidErr);
  }
  for (int i = dim; i--;) {
    if (VecIsEqual(facoid->_axis[i], clone->_axis[i]) == false) {
      ShapoidErr->_type = PBErrTypeUnitTestFailed;
      sprintf(ShapoidErr->_msg, "ShapoidClone failed");
      PBErrCatch(ShapoidErr);
    }
  }
  if (ShapoidIsEqual(facoid, clone) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidIsEqual failed");
    PBErrCatch(ShapoidErr);
  }
  clone->_type = ShapoidTypePyramidoid;
  if (ShapoidIsEqual(facoid, clone) == true) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidIsEqual failed");
    PBErrCatch(ShapoidErr);
  }
  clone->_type = facoid->_type;
  clone->_dim = dim + 1;
  if (ShapoidIsEqual(facoid, clone) == true) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidIsEqual failed");
    PBErrCatch(ShapoidErr);
  }
  clone->_dim = facoid->_dim;
  VecSet(clone->_pos, 0, 1.0);
  if (ShapoidIsEqual(facoid, clone) == true) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidIsEqual failed");
    PBErrCatch(ShapoidErr);
  }
  VecSet(clone->_pos, 0, 0.0);
  VecSet(clone->_axis[0], 0, 2.0);
  if (ShapoidIsEqual(facoid, clone) == true) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidIsEqual failed");
    PBErrCatch(ShapoidErr);
  }
  VecSet(clone->_axis[0], 0, 1.0);
  ShapoidFree(&facoid);
  if (facoid != NULL) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidFree failed");
    PBErrCatch(ShapoidErr);
  }
  ShapoidFree(&clone);
  printf("UnitTestCreateCloneIsEqualFree OK\n");
}

void UnitTestLoadSavePrint() {
  int dim = 3;
  Shapoid *facoid = FacoidCreate(dim);
  FILE *file = fopen("./facoid.txt", "w");
  if (ShapoidSave(facoid, file) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidSave failed");
    PBErrCatch(ShapoidErr);
  }
  fclose(file);
  file = fopen("./facoid.txt", "r");
  Shapoid *load = NULL;
  if (ShapoidLoad(&load, file) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidLoad failed");
    PBErrCatch(ShapoidErr);
  }
  fclose(file);
  if (ShapoidIsEqual(facoid, load) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidLoad/Save failed");
    PBErrCatch(ShapoidErr);
  }
  ShapoidPrintln(facoid, stdout);
  ShapoidFree(&facoid);
  ShapoidFree(&load);
  printf("UnitTestLoadSavePrint OK\n");
}

void UnitTestGetSetTypeDimPosAxis() {
  int dim = 3;
  Shapoid *facoid = FacoidCreate(dim);
  Shapoid *pyramidoid = PyramidoidCreate(dim);
  Shapoid *spheroid = SpheroidCreate(dim);
  if (ShapoidGetType(facoid) != ShapoidTypeFacoid ||
    ShapoidGetType(pyramidoid) != ShapoidTypePyramidoid ||
    ShapoidGetType(spheroid) != ShapoidTypeSpheroid) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidGetType failed");
    PBErrCatch(ShapoidErr);
  }
  if (ShapoidGetDim(facoid) != dim) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidGetDim failed");
    PBErrCatch(ShapoidErr);
  }
  VecFloat *v = VecFloatCreate(dim);
  VecFloat *u = ShapoidGetPos(facoid);
  if (VecIsEqual(v, u) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidGetPos failed");
    PBErrCatch(ShapoidErr);
  }
  VecFree(&u);
  for (int i = dim; i--;) {
    u = ShapoidGetAxis(facoid, i);
    for (int j = dim; j--;)
      if ((i == j && ISEQUALF(VecGet(u, j), 1.0) == false) ||
        (i != j && ISEQUALF(VecGet(u, j), 0.0) == false)) {
        ShapoidErr->_type = PBErrTypeUnitTestFailed;
        sprintf(ShapoidErr->_msg, "ShapoidGetAxis failed");
        PBErrCatch(ShapoidErr);
      }
    VecFree(&u);
  }
  for (int i = dim; i--;)
    VecSet(v, i, (float)i);
  ShapoidSetPos(facoid, v);
  if (VecIsEqual(v, facoid->_pos) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidSetPos failed");
    PBErrCatch(ShapoidErr);
  }
  for (int i = dim; i--;) {
    VecSetNull(v);
    VecSet(v, i, 2.0);
    ShapoidSetAxis(facoid, i, v);
    if (VecIsEqual(v, facoid->_axis[i]) == false) {
      ShapoidErr->_type = PBErrTypeUnitTestFailed;
      sprintf(ShapoidErr->_msg, "ShapoidSetAxis failed");
      PBErrCatch(ShapoidErr);
    }
  }
  VecFree(&v);
  ShapoidFree(&facoid);
  ShapoidFree(&pyramidoid);
  ShapoidFree(&spheroid);
  printf("UnitTestGetSetTypeDimPosAxis OK\n");
```

```
}

void UnitTestTranslateScaleGrowRotate() {
  int dim = 2;
  Shapoid *facoid = FacoidCreate(dim);
  VecFloat *v = VecFloatCreate(dim);
  for (int i = dim; i--;)
    VecSet(v, i, 1.0);
  ShapoidTranslate(facoid, v);
  if (VecIsEqual(v, facoid->_pos) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidTranslate failed");
    PBErrCatch(ShapoidErr);
  }
  float scale = 2.0;
  ShapoidScale(facoid, scale);
  VecSetNull(v);
  VecSetNull(facoid->_pos);
  if (VecIsEqual(v, facoid->_pos) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidScaleScalar failed");
    PBErrCatch(ShapoidErr);
  }
  for (int i = dim; i--;) {
    for (int j = dim; j--;)
      if (i == j)
        VecSet(v, j, scale);
      else
        VecSet(v, j, 0.0);
    if (VecIsEqual(v, facoid->_axis[i]) == false) {
      ShapoidErr->_type = PBErrTypeUnitTestFailed;
      sprintf(ShapoidErr->_msg, "ShapoidScaleScalar failed");
      PBErrCatch(ShapoidErr);
    }
  }
  for (int i = dim; i--;)
    VecSet(v, i, 1.0 + (float)i);
  ShapoidScale(facoid, v);
  VecSetNull(v);
  if (VecIsEqual(v, facoid->_pos) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidScaleVector failed");
    PBErrCatch(ShapoidErr);
  }
  for (int i = dim; i--;) {
    for (int j = dim; j--;)
      if (i == j)
        VecSet(v, j, scale * (1.0 + (float)i));
      else
        VecSet(v, j, 0.0);
    if (VecIsEqual(v, facoid->_axis[i]) == false) {
      ShapoidErr->_type = PBErrTypeUnitTestFailed;
      sprintf(ShapoidErr->_msg, "ShapoidScaleVector failed");
      PBErrCatch(ShapoidErr);
    }
  }
  ShapoidFree(&facoid);
  facoid = FacoidCreate(dim);
  scale = 2.0;
  ShapoidGrow(facoid, scale);
  for (int i = dim; i--;)
    VecSet(v, i, -0.5);
```

```
if (VecIsEqual(v, facoid->_pos) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGrowScalar failed");
  PBErrCatch(ShapoidErr);
}
for (int i = dim; i--;) {
  for (int j = dim; j--;)
    if (i == j)
      VecSet(v, j, scale);
    else
      VecSet(v, j, 0.0);
  if (VecIsEqual(v, facoid->_axis[i]) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidGrowScalar failed");
    PBErrCatch(ShapoidErr);
  }
}
Shapoid *pyramidoid = PyramidoidCreate(dim);
VecFloat *centerA = ShapoidGetCenter(pyramidoid);
ShapoidGrow(pyramidoid, scale);
VecFloat *centerB = ShapoidGetCenter(pyramidoid);
if (VecIsEqual(centerA, centerB) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGrowScalar failed");
  PBErrCatch(ShapoidErr);
}
for (int i = dim; i--;) {
  for (int j = dim; j--;)
    if (i == j)
      VecSet(v, j, scale);
    else
      VecSet(v, j, 0.0);
  if (VecIsEqual(v, pyramidoid->_axis[i]) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidGrowScalar failed");
    PBErrCatch(ShapoidErr);
  }
}
VecFree(&centerA);
VecFree(&centerB);
Shapoid *spheroid = SpheroidCreate(dim);
ShapoidGrow(spheroid, scale);
VecSetNull(v);
if (VecIsEqual(v, spheroid->_pos) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGrowScalar failed");
  PBErrCatch(ShapoidErr);
}
for (int i = dim; i--;) {
  for (int j = dim; j--;)
    if (i == j)
      VecSet(v, j, scale);
    else
      VecSet(v, j, 0.0);
  if (VecIsEqual(v, spheroid->_axis[i]) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidGrowScalar failed");
    PBErrCatch(ShapoidErr);
  }
}
VecFloat *scalev = VecFloatCreate(dim);
for (int i = dim; i--;)
```

```
  VecSet(scalev, i, 1.0 + (float)i);
ShapoidFree(&facoid);
ShapoidFree(&pyramidoid);
ShapoidFree(&spheroid);
facoid = FacoidCreate(dim);
ShapoidGrow(facoid, scalev);
float pa[2] = {0.000,-0.500};
for (int i = dim; i--;)
  VecSet(v, i, pa[i]);
if (VecIsEqual(v, facoid->_pos) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGrowVector failed");
  PBErrCatch(ShapoidErr);
}
for (int i = dim; i--;) {
  for (int j = dim; j--;)
    if (i == j)
      VecSet(v, j, VecGet(scalev, i));
    else
      VecSet(v, j, 0.0);
  if (VecIsEqual(v, facoid->_axis[i]) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidGrowVector failed");
    PBErrCatch(ShapoidErr);
  }
}
pyramidoid = PyramidoidCreate(dim);
centerA = ShapoidGetCenter(pyramidoid);
ShapoidGrow(pyramidoid, scalev);
centerB = ShapoidGetCenter(pyramidoid);
if (VecIsEqual(centerA, centerB) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGrowVector failed");
  PBErrCatch(ShapoidErr);
}
for (int i = dim; i--;) {
  for (int j = dim; j--;)
    if (i == j)
      VecSet(v, j, VecGet(scalev, i));
    else
      VecSet(v, j, 0.0);
  if (VecIsEqual(v, pyramidoid->_axis[i]) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidGrowVector failed");
    PBErrCatch(ShapoidErr);
  }
}
VecFree(&centerA);
VecFree(&centerB);
spheroid = SpheroidCreate(dim);
ShapoidGrow(spheroid, scalev);
VecSetNull(v);
if (VecIsEqual(v, spheroid->_pos) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGrowVector failed");
  PBErrCatch(ShapoidErr);
}
for (int i = dim; i--;) {
  for (int j = dim; j--;)
    if (i == j)
      VecSet(v, j, VecGet(scalev, i));
    else
```

```
        VecSet(v, j, 0.0);
    if (VecIsEqual(v, spheroid->_axis[i]) == false) {
      ShapoidErr->_type = PBErrTypeUnitTestFailed;
      sprintf(ShapoidErr->_msg, "ShapoidGrowVector failed");
      PBErrCatch(ShapoidErr);
    }
  }
  VecFree(&scalev);
  ShapoidFree(&facoid);
  ShapoidFree(&pyramidoid);
  ShapoidFree(&spheroid);
  facoid = FacoidCreate(dim);
  pyramidoid = PyramidoidCreate(dim);
  spheroid = SpheroidCreate(dim);
  float theta = PBMATH_HALFPI;
  ShapoidRotate2D(facoid, theta);
  float pb[2] = {1.0, 0.0};
  float pc[2] = {0.0, 1.0};
  float pd[2] = {-1.0, 0.0};
  for (int i = dim; i--;) {
    if (ISEQUALF(VecGet(facoid->_pos, i), pb[i]) == false ||
      ISEQUALF(VecGet(facoid->_axis[0], i), pc[i]) == false ||
      ISEQUALF(VecGet(facoid->_axis[1], i), pd[i]) == false) {
      ShapoidErr->_type = PBErrTypeUnitTestFailed;
      sprintf(ShapoidErr->_msg, "ShapoidRotate2D failed");
      PBErrCatch(ShapoidErr);
    }
  }
  ShapoidRotate2D(pyramidoid, theta);
  float pe[2] = {0.6666667, 0.0};
  float pf[2] = {0.0, 1.0};
  float pg[2] = {-1.0, 0.0};
  for (int i = dim; i--;) {
    if (ISEQUALF(VecGet(pyramidoid->_pos, i), pe[i]) == false ||
      ISEQUALF(VecGet(pyramidoid->_axis[0], i), pf[i]) == false ||
      ISEQUALF(VecGet(pyramidoid->_axis[1], i), pg[i]) == false) {
      ShapoidErr->_type = PBErrTypeUnitTestFailed;
      sprintf(ShapoidErr->_msg, "ShapoidRotate2D failed");
      PBErrCatch(ShapoidErr);
    }
  }
  ShapoidRotate2D(spheroid, theta);
  float ph[2] = {0.0, 0.0};
  float pi[2] = {0.0, 1.0};
  float pj[2] = {-1.0, 0.0};
  for (int i = dim; i--;) {
    if (ISEQUALF(VecGet(spheroid->_pos, i), ph[i]) == false ||
      ISEQUALF(VecGet(spheroid->_axis[0], i), pi[i]) == false ||
      ISEQUALF(VecGet(spheroid->_axis[1], i), pj[i]) == false) {
      ShapoidErr->_type = PBErrTypeUnitTestFailed;
      sprintf(ShapoidErr->_msg, "ShapoidRotate2D failed");
      PBErrCatch(ShapoidErr);
    }
  }
  VecFree(&v);
  ShapoidFree(&facoid);
  ShapoidFree(&pyramidoid);
  ShapoidFree(&spheroid);
  printf("UnitTestTranslateScaleGrowRotate OK\n");
}

void UnitTestImportExportCoordIsPosInside() {
```

```
int dim = 2;
Shapoid *facoid = FacoidCreate(dim);
Shapoid *pyramidoid = PyramidoidCreate(dim);
Shapoid *spheroid = SpheroidCreate(dim);
VecFloat *v = VecFloatCreate(dim);
for (int i = dim; i--;)
  VecSet(v, i, 1.0 + (float)i);
ShapoidTranslate(facoid, v);
ShapoidTranslate(pyramidoid, v);
ShapoidTranslate(spheroid, v);
float scale = -2.0;
ShapoidScale(facoid, scale);
ShapoidScale(pyramidoid, scale);
ShapoidScale(spheroid, scale);
int nbTest = 100;
srandom(RANDOMSEED);
for (int iTest = nbTest; iTest--;) {
  VecFloat *posReal = VecFloatCreate(dim);
  for (int i = dim; i--;)
    VecSet(posReal, i, (rnd() - 0.5) * 10.0);
  VecFloat *posShapoidA = ShapoidImportCoord(facoid, posReal);
  bool isInside = ShapoidIsPosInside(facoid, posReal);
  if (VecGet(posShapoidA, 0) >= 0.0 &&
    VecGet(posShapoidA, 0) <= 1.0 &&
    VecGet(posShapoidA, 1) >= 0.0 &&
    VecGet(posShapoidA, 1) <= 1.0) {
    if (isInside == false) {
      ShapoidErr->_type = PBErrTypeUnitTestFailed;
      sprintf(ShapoidErr->_msg, "ShapoidIsPosInside failed");
      PBErrCatch(ShapoidErr);
    }
  } else {
    if (isInside == true) {
      ShapoidErr->_type = PBErrTypeUnitTestFailed;
      sprintf(ShapoidErr->_msg, "ShapoidIsPosInside failed");
      PBErrCatch(ShapoidErr);
    }
  }
  VecOp(posShapoidA, scale, v, 1.0);
  if (VecIsEqual(posReal, posShapoidA) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidImportCoord failed");
    PBErrCatch(ShapoidErr);
  }
  VecFree(&posShapoidA);
  VecFloat *posShapoidB = ShapoidImportCoord(pyramidoid, posReal);
  isInside = ShapoidIsPosInside(pyramidoid, posReal);
  if (VecGet(posShapoidB, 0) >= 0.0 &&
    VecGet(posShapoidB, 0) <= 1.0 &&
    VecGet(posShapoidB, 1) >= 0.0 &&
    VecGet(posShapoidB, 1) <= 1.0 &&
    VecGet(posShapoidB, 0) + VecGet(posShapoidB, 1) <= 1.0) {
    if (isInside == false) {
      ShapoidErr->_type = PBErrTypeUnitTestFailed;
      sprintf(ShapoidErr->_msg, "ShapoidIsPosInside failed");
      PBErrCatch(ShapoidErr);
    }
  } else {
    if (isInside == true) {
      ShapoidErr->_type = PBErrTypeUnitTestFailed;
      sprintf(ShapoidErr->_msg, "ShapoidIsPosInside failed");
      PBErrCatch(ShapoidErr);
    }
  }
  VecOp(posShapoidB, scale, v, 1.0);
  if (VecIsEqual(posReal, posShapoidB) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidImportCoord failed");
    PBErrCatch(ShapoidErr);
  }
  VecFree(&posShapoidB);
  VecFloat *posShapoidC = ShapoidImportCoord(spheroid, posReal);
  isInside = ShapoidIsPosInside(spheroid, posReal);
  if (VecGet(posShapoidC, 0) >= -0.5 &&
    VecGet(posShapoidC, 0) <= 0.5 &&
    VecGet(posShapoidC, 1) >= -0.5 &&
    VecGet(posShapoidC, 1) <= 0.5 &&
    pow(VecGet(posShapoidC, 0), 2.0) +
    pow(VecGet(posShapoidC, 1), 2.0) <= 0.25) {
    if (isInside == false) {
      ShapoidErr->_type = PBErrTypeUnitTestFailed;
      sprintf(ShapoidErr->_msg, "ShapoidIsPosInside failed");
      PBErrCatch(ShapoidErr);
    }
  } else {
    if (isInside == true) {
      ShapoidErr->_type = PBErrTypeUnitTestFailed;
      sprintf(ShapoidErr->_msg, "ShapoidIsPosInside failed");
      PBErrCatch(ShapoidErr);
    }
  }
  VecOp(posShapoidC, scale, v, 1.0);
  if (VecIsEqual(posReal, posShapoidC) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidImportCoord failed");
    PBErrCatch(ShapoidErr);
  }
  VecFree(&posShapoidC);
  VecFree(&posReal);
}
for (int iTest = nbTest; iTest--;) {
  VecFloat *posShapoid = VecFloatCreate(dim);
  for (int i = dim; i--;)
    VecSet(posShapoid, i, (rnd() - 0.5) * 10.0);
  VecFloat *posRealA = ShapoidExportCoord(facoid, posShapoid);
  VecOp(posRealA, 1.0, v, -1.0);
  VecScale(posRealA, 1.0 / scale);
  if (VecIsEqual(posRealA, posShapoid) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidExportCoord failed");
    PBErrCatch(ShapoidErr);
  }
  VecFree(&posRealA);
  VecFloat *posRealB = ShapoidExportCoord(pyramidoid, posShapoid);
  VecOp(posRealB, 1.0, v, -1.0);
  VecScale(posRealB, 1.0 / scale);
  if (VecIsEqual(posRealB, posShapoid) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidExportCoord failed");
    PBErrCatch(ShapoidErr);
  }
  VecFree(&posRealB);
  VecFloat *posRealC = ShapoidExportCoord(facoid, posShapoid);
  VecOp(posRealC, 1.0, v, -1.0);
  VecScale(posRealC, 1.0 / scale);
  if (VecIsEqual(posRealC, posShapoid) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidExportCoord failed");
    PBErrCatch(ShapoidErr);
  }
  VecFree(&posRealC);
  VecFree(&posShapoid);
}
VecFree(&v);
ShapoidFree(&facoid);
ShapoidFree(&pyramidoid);
ShapoidFree(&spheroid);
printf("UnitTestImportExportCoordIsPosInside OK\n");
}

void UnitTestGetBoundingBox() {
int dim = 2;
Shapoid *facoid = FacoidCreate(dim);
Shapoid *pyramidoid = PyramidoidCreate(dim);
Shapoid *spheroid = SpheroidCreate(dim);
VecFloat *v = VecFloatCreate(dim);
for (int i = dim; i--;)
  VecSet(v, i, 1.0 + (float)i);
ShapoidTranslate(facoid, v);
ShapoidTranslate(pyramidoid, v);
ShapoidTranslate(spheroid, v);
float scale = -2.0;
ShapoidScale(facoid, scale);
ShapoidScale(pyramidoid, scale);
ShapoidScale(spheroid, scale);
float theta = PBMATH_QUARTERPI;
ShapoidRotate2D(facoid, theta);
ShapoidRotate2D(pyramidoid, theta);
ShapoidRotate2D(spheroid, theta);
Shapoid *boundA = ShapoidGetBoundingBox(facoid);
float pa[2] = {-1.414214, -0.414213};
float pb[2] = {2.828427, 0.0};
float pc[2] = {0.0, 2.828427};
for (int i = dim; i--;) {
  if (ISEQUALF(VecGet(boundA->_pos, i), pa[i]) == false ||
    ISEQUALF(VecGet(boundA->_axis[0], i), pb[i]) == false ||
    ISEQUALF(VecGet(boundA->_axis[1], i), pc[i]) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidGetBoundingBox failed");
    PBErrCatch(ShapoidErr);
  }
}
Shapoid *boundB = ShapoidGetBoundingBox(pyramidoid);
float pd[2] = {-1.414214, -1.4142143};
float pe[2] = {2.828427, 0.0};
float pf[2] = {0.0, 3.690356};
for (int i = dim; i--;) {
  if (ISEQUALF(VecGet(boundB->_pos, i), pd[i]) == false ||
    ISEQUALF(VecGet(boundB->_axis[0], i), pe[i]) == false ||
    ISEQUALF(VecGet(boundB->_axis[1], i), pf[i]) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidGetBoundingBox failed");
    PBErrCatch(ShapoidErr);
  }
}
Shapoid *boundC = ShapoidGetBoundingBox(spheroid);
float pg[2] = {-0.414214, 0.585786};
float ph[2] = {2.828427, 0.0};
float pi[2] = {0.0, 2.828427};
for (int i = dim; i--;) {
  if (ISEQUALF(VecGet(boundC->_pos, i), pg[i]) == false ||
    ISEQUALF(VecGet(boundC->_axis[0], i), ph[i]) == false ||
    ISEQUALF(VecGet(boundC->_axis[1], i), pi[i]) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidGetBoundingBox failed");
    PBErrCatch(ShapoidErr);
  }
}
GSet set = GSetCreateStatic();
GSetPush(&set, facoid);
GSetPush(&set, pyramidoid);
GSetPush(&set, spheroid);
Shapoid *boundD = ShapoidGetBoundingBox(&set);
float pj[2] = {-1.414214, -1.4142143};
float pk[2] = {3.828427, 0.0};
float pl[2] = {0.0, 4.828427};
for (int i = dim; i--;) {
  if (ISEQUALF(VecGet(boundD->_pos, i), pj[i]) == false ||
    ISEQUALF(VecGet(boundD->_axis[0], i), pk[i]) == false ||
    ISEQUALF(VecGet(boundD->_axis[1], i), pl[i]) == false) {
    ShapoidErr->_type = PBErrTypeUnitTestFailed;
    sprintf(ShapoidErr->_msg, "ShapoidGetBoundingBox failed");
    PBErrCatch(ShapoidErr);
  }
}
GSetFlush(&set);
ShapoidFree(&boundA);
ShapoidFree(&boundB);
ShapoidFree(&boundC);
ShapoidFree(&boundD);
VecFree(&v);
ShapoidFree(&facoid);
ShapoidFree(&pyramidoid);
ShapoidFree(&spheroid);
printf("UnitTestGetBoundingBox OK\n");
}

void UnitTestGetPosDepthCenterCoverage() {
int dim = 2;
Shapoid *facoid = FacoidCreate(dim);
Shapoid *pyramidoid = PyramidoidCreate(dim);
Shapoid *spheroid = SpheroidCreate(dim);
VecFloat *center = ShapoidGetCenter(facoid);
if (ISEQUALF(VecGet(center, 0), 0.5) == false ||
  ISEQUALF(VecGet(center, 1), 0.5) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGetCenter failed");
  PBErrCatch(ShapoidErr);
}
VecFree(&center);
center = ShapoidGetCenter(pyramidoid);
if (ISEQUALF(VecGet(center, 0), 0.333333) == false ||
  ISEQUALF(VecGet(center, 1), 0.333333) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGetCenter failed");
  PBErrCatch(ShapoidErr);
}
VecFree(&center);
```

```
center = ShapoidGetCenter(spheroid);
if (ISEQUALF(VecGet(center, 0), 0.0) == false ||
  ISEQUALF(VecGet(center, 1), 0.0) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGetCenter failed");
  PBErrCatch(ShapoidErr);
}
VecFree(&center);
float coverage = ShapoidGetCoverageDelta(facoid, pyramidoid, 0.001);
if (ISEQUALF(coverage, 1.0) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGetCoverage failed");
  PBErrCatch(ShapoidErr);
}
coverage = ShapoidGetCoverageDelta(pyramidoid, facoid, 0.001);
if (ISEQUALF(coverage, 0.500499) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGetCoverage failed");
  PBErrCatch(ShapoidErr);
}
coverage = ShapoidGetCoverageDelta(pyramidoid, spheroid, 0.001);
if (ISEQUALF(coverage, 0.24937) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGetCoverage failed");
  PBErrCatch(ShapoidErr);
}
coverage = ShapoidGetCoverageDelta(spheroid, pyramidoid, 0.001);
if (ISEQUALF(coverage, 0.39251) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGetCoverage failed");
  PBErrCatch(ShapoidErr);
}
coverage = ShapoidGetCoverageDelta(facoid, spheroid, 0.001);
if (ISEQUALF(coverage, 0.24937) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGetCoverage failed");
  PBErrCatch(ShapoidErr);
}
coverage = ShapoidGetCoverageDelta(spheroid, facoid, 0.001);
if (ISEQUALF(coverage, 0.196451) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGetCoverage failed");
  PBErrCatch(ShapoidErr);
}
VecFloat2D pos = VecFloatCreateStatic2D();
VecSet(&pos, 0, 0.333333); VecSet(&pos, 1, 0.333333);
float depth = ShapoidGetPosDepth(facoid, (VecFloat*)&pos);
if (ISEQUALF(depth, 0.790123) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGetPosDepth failed");
  PBErrCatch(ShapoidErr);
}
depth = ShapoidGetPosDepth(pyramidoid, (VecFloat*)&pos);
if (ISEQUALF(depth, 1.0) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGetPosDepth failed");
  PBErrCatch(ShapoidErr);
}
depth = ShapoidGetPosDepth(spheroid, (VecFloat*)&pos);
if (ISEQUALF(depth, 0.057192) == false) {
  ShapoidErr->_type = PBErrTypeUnitTestFailed;
  sprintf(ShapoidErr->_msg, "ShapoidGetPosDepth failed");
```

```
    PBErrCatch(ShapoidErr);
  }
  ShapoidFree(&facoid);
  ShapoidFree(&pyramidoid);
  ShapoidFree(&spheroid);
  printf("UnitTestGetPosDepthCenterCoverage OK\n");
}

void UnitTestAll() {
  UnitTestCreateCloneIsEqualFree();
  UnitTestLoadSavePrint();
  UnitTestGetSetTypeDimPosAxis();
  UnitTestTranslateScaleGrowRotate();
  UnitTestImportExportCoordIsPosInside();
  UnitTestGetBoundingBox();
  UnitTestGetPosDepthCenterCoverage();
  printf("UnitTestAll OK\n");
}

int main() {
  UnitTestAll();
  // Return success code
  return 0;
}
```

6 Unit tests output

```
UnitTestCreateCloneIsEqualFree OK
Type: Facoid
Dim: 3
Pos: <0.000,0.000,0.000>
Axis(0): <1.000,0.000,0.000>
Axis(1): <0.000,1.000,0.000>
Axis(2): <0.000,0.000,1.000>
UnitTestLoadSavePrint OK
UnitTestGetSetTypeDimPosAxis OK
UnitTestTranslateScaleGrowRotate OK
UnitTestImportExportCoordIsPosInside OK
UnitTestGetBoundingBox OK
UnitTestGetPosDepthCenterCoverage OK
UnitTestAll OK
```

facoid.txt

```
3 0
3 0.000000 0.000000 0.000000
3 1.000000 0.000000 0.000000
3 0.000000 1.000000 0.000000
3 0.000000 0.000000 1.000000
```