This paper has been published in : *Graphics Gems, vol 5, p249-256, Academic Press*

Generic Implementation of Axial Deformation Techniques

Carole Blanc

LaBRI,[1] *351 cours de la libération, 33405 Talence (France)*
blanc@labri.u-bordeaux.fr

Introduction

Global deformation techniques were first introduced to extend the set of primitives that may be used in constructive solid modeling [1, 5]. In fact, these techniques are more general and have been consequently adapted to tessellated surfaces or parametric patches as well. This gem proposes a generic implementation of several of these global deformation techniques. The term "generic" focuses on the fact that the implementation depends neither on a given geometric model for surfaces nor on a specific data structures for internal representation.

The principle of global deformation is to map each point (x, y, z) in the Euclidian space $\mathbb{R}^3$ onto another point (x', y', z') under a deformation function $f(x, y, z, a, b, c, ...)$. The additional parameters $(a, b, c, ...)$ used by the deformation function are either constants or values returned by *shape functions*. Moreover, according to the way these shape functions are defined, global deformation techniques may be further classified as procedural or interactive.

There is a subset of global deformation techniques for which the shape functions operate upon only one coordinate (typically z) of the initial point P. For this and related reasons [4] such techniques are called *axial deformation techniques* in what follows. Although being relatively specific compared to more general techniques such as FFD [5] or EFFD [3], axial deformations have been found useful in many computer graphics applications; for instance, all the procedural techniques proposed in Barr's original work on global deformations [1] were axial deformations.

Description

The general mapping function for axial deformations may be expressed by:

$$(x', y', z') = f(x, y, z, a(z), b(z), c(z), ...) \tag{1}$$

The axial deformations proposed here are defined with a simplified form of Equation (1) which uses only two additional parameters :

$$(x', y', z') = f(x, y, z, s(z), a) \tag{2}$$

where $s(z) : \mathbb{R} \rightarrow [0, 1]$ is the shape function and $a \in \mathbb{R}$ is the amplitude of the deformation. The role of both parameters may be understand intuitively: when the shape function $s(z)$ is null for a given point (x, y, z), it means that this point will be unchanged by the deformation; similarly, when the amplitude a is null, the whole object remains undeformed.

In the generic implementation proposed here, each deformation routine acts only on points: it takes the coordinates of a point on the original object, computes its displacement according to the deformation function, and finally returns the new coordinates of the point. A complete implementation of axial deformations should include some interactive tools for defining the shape function. Such a tool lies outside the scope of this gem; but many interesting procedural deformations may be obtained by using general-purpose functions, such as the *wave generators* described elsewhere in this volume [6].

[1] Laboratoire Bordelais de Recherche en Informatique (*Université Bordeaux I* and *Centre National de la Recherche Scientifique*). The present work is also granted by the *Conseil Régional d'Aquitaine*.

Six different axial deformations are provided (**pinch, taper, mould, twist, shear, bend**) which differ only by the function f that is applied on each point. Figure 1 shows several objects that may be obtain with the six deformation operators, starting from a parallelpiped. In fact, two kinds of operators are used here : the first ones (pinch, shear, bend) take the Cartesian coordinates (x, y, z) of the point in the local frame, whereas the second one (taper, mould, twist) act on the cylindrical coordinates (r, θ, z). The following lines give an overview of the work that is done by each operator :

pinch: The x coordinate is scaled according to a and $s(z)$
taper: The r coordinate is scaled according to a and $s(z)$
mould: The r coordinate is scaled according to a and $s(\theta)$
twist: The θ coordinate is scaled according to a and $s(z)$
shear: The z axis is translated according to a and $s(z)$
bend: The z axis is rotated according to a and $s(z)$

For each deformation technique, two routines are provided. The first (`local_*`) assumes that the coordinates of the incoming point are already expressed in the local frame where the deformation is defined (i.e. the frame for which the z coordinate is used by the shape function). The second one (`world_*`) takes the coordinates of the point in the world frame and returns the coordinates in the same frame. For this, it needs an additional parameter defining the local frame and performs the deformation within this frame. The implementation uses the toolbox of macro functions defined in the present volume [7] which provides the frame conversion routines.

Finally, note that a *unit-cube frame* [2] may be employed instead of the traditional frame having unit vectors, because it facilitates the deformation of objects independently of their position, orientation and size.

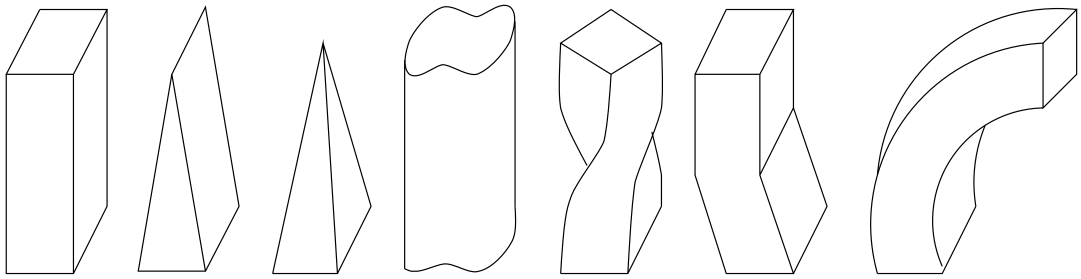

Figure 1: *(a) Original object (b) pinch (c) taper (d) mould (e) twist (f) shear (g) bend*

Source Files

```
/* ------------------------------------------------------------------------- *\
   AXD.H : Carole Blanc (4 June 1994)

   This package provides an implementation of 6 different algorithms
   for doing axial deformations.

   "Generic Implementation of Axial Deformation Techniques"
   in Graphics Gems V (edited by A. Paeth), Academic Press
\* ------------------------------------------------------------------------- */

#ifndef _AXD_
#define  _AXD_

/*
** This package uses the "Toolbox of Macros Functions for Computer Graphics"
** which provides files : tool.h, real.h, uint.h, sint.h, vec?.h and mat?.h
*/
```

```
#include "real.h"

typedef real (*shape) (real);

extern void local_pinch (rv3 *Point, shape Shape, real Ampli);
extern void world_pinch (rv3 *Point, frame3 Frame, shape Shape, real Ampli);
extern void local_taper (rv3 *Point, shape Shape, real Ampli);
extern void world_taper (rv3 *Point, frame3 Frame, shape Shape, real Ampli);
extern void local_mould (rv3 *Point, shape Shape, real Ampli);
extern void world_mould (rv3 *Point, frame3 Frame, shape Shape, real Ampli);
extern void local_twist (rv3 *Point, shape Shape, real Ampli);
extern void world_twist (rv3 *Point, frame3 Frame, shape Shape, real Ampli);
extern void local_shear (rv3 *Point, shape Shape, real Ampli);
extern void world_shear (rv3 *Point, frame3 Frame, shape Shape, real Ampli);
extern void local_bend  (rv3 *Point, shape Shape, real Ampli);
extern void world_bend  (rv3 *Point, frame3 Frame, shape Shape, real Ampli);

#endif

/* ------------------------------------------------------------------------- *\
   AXD.C : Carole Blanc (4 June 1994)

   This package provides an implementation of 6 different algorithms
   for doing axial deformations.

   "Generic Implementation of Axial Deformation Techniques"
   in Graphics Gems V (edited by A. Paeth), Academic Press
\* ------------------------------------------------------------------------- */

#include "axd.h"
#include "mat3.h"

/*
** Each "local_*" routines inputs/outputs the following arguments
**
** Input:  Point = coordinates of the point in the local frame
**         Shape = shape function of the deformation
**         Ampli = amplitude of the deformation
** Output: Point = coordinates of the deformed point in the local frame
**
** Each "world_*" routines inputs/outputs the following arguments
**
** Input:  Point = coordinates of the point in the world frame
**         Frame = local frame in which the deformation is applied
**         Shape = shape function of the deformation
**         Ampli = amplitude of the deformation
** Output: Point = coordinates of the deformed point in the world frame
**
** Note: The "Frame" argument must be initialized by MAKE_FRAME3 (see "mat3.h")
*/

/*
** pinch : Scale the x coordinate of the object according to z
*/

void local_pinch (realvec3 *Point, shape Shape, real Ampli)
{
```

```
  Point->x *= 1.0 - Ampli * Shape (Point->z);
}

void world_pinch (realvec3 *Point, frame3 Frame, shape Shape, real Ampli)
{
  LOCAL_FRAME3 (Point, Frame);
  local_pinch  (Point, Shape, Ampli);
  WORLD_FRAME3 (Point, Frame);
}

/*
** taper : Scale the polar radius of the object according to z
*/

void local_taper (realvec3 *Point, shape Shape, real Ampli)
{
  register real Tmp;

  Tmp = 1.0 - Ampli * Shape (Point->z); Point->x *= Tmp; Point->y *= Tmp;
}

void world_taper (realvec3 *Point, frame3 Frame, shape Shape, real Ampli)
{
  LOCAL_FRAME3 (Point, Frame);
  local_taper  (Point, Shape, Ampli);
  WORLD_FRAME3 (Point, Frame);
}

/*
** mould : Scale the polar radius of the object according to the polar angle
*/

void local_mould (realvec3 *Point, shape Shape, real Ampli)
{
  register real Tmp;

  Tmp = atan2 (Point->y, Point->x) / PI;
  Tmp = 1.0 - Ampli * Shape (Tmp); Point->x *= Tmp; Point->y *= Tmp;
}

void world_mould (realvec3 *Point, frame3 Frame, shape Shape, real Ampli)
{
  LOCAL_FRAME3 (Point, Frame);
  local_mould  (Point, Shape, Ampli);
  WORLD_FRAME3 (Point, Frame);
}

/*
** twist : Scale the polar angle of the object according to z
*/

void local_twist (realvec3 *Point, shape Shape, real Ampli)
{
  register real Tmp, Cos, Sin;

  Tmp = PI * Ampli * Shape (Point->z);
  Cos = cos (Tmp); Sin = sin (Tmp); Tmp = Point->x;
```

```
  Point->x = Cos * Tmp - Sin * Point->y;
  Point->y = Sin * Tmp + Cos * Point->y;
}

void world_twist (realvec3 *Point, frame3 Frame, shape Shape, real Ampli)
{
  LOCAL_FRAME3 (Point, Frame);
  local_twist  (Point, Shape, Ampli);
  WORLD_FRAME3 (Point, Frame);
}

/*
** shear : Translate the z axis of the object along x according to z
*/

void local_shear (realvec3 *Point, shape Shape, real Ampli)
{
  Point->x += Ampli * Shape (Point->z);
}

void world_shear (realvec3 *Point, frame3 Frame, shape Shape, real Ampli)
{
  LOCAL_FRAME3 (Point, Frame);
  local_shear  (Point, Shape, Ampli);
  WORLD_FRAME3 (Point, Frame);
}

/*
** bend : Rotate the z axis of the object around y according to z
*/

void local_bend (realvec3 *Point, shape Shape, real Ampli)
{
  register real Tmp, Cos, Sin;

  Tmp = PI * Ampli * Shape (Point->z);
  Cos = cos (Tmp); Sin = sin (Tmp); Tmp = Point->z;
  Point->z = Cos * Tmp - Sin * Point->x;
  Point->x = Sin * Tmp + Cos * Point->x;
}

void world_bend (realvec3 *Point, frame3 Frame, shape Shape, real Ampli)
{
  LOCAL_FRAME3 (Point, Frame);
  local_bend   (Point, Shape, Ampli);
  WORLD_FRAME3 (Point, Frame);
}
```

Bibliography

[1] Barr, A., *Global and local deformations of solid primitives*, Computer Graphics, v18, n3, p21-30, 1984.

[2] Blanc, C. and Schlick, C., *Easy transformations between cartesian, cylindrical and spherical coordinates*, Technical Report 832-94, LaBRI, Univ Bordeaux I (France) , 1994.

[3] Coquillart, S., *Extended Free Form Deformations: A Sculpturing Tool for 3D Geometric Modeling*, Computer Graphics, v24, n4, p187-196, 1990.

[4] Lazarus, F. and Coquillart, S. and Jancene, P., *Interactive axial deformations (in French)*, Proc. of Groplan 92 (Nantes, France), p117–124, 1992.

[5] Sederberg, T. and Parry, S., *Free-form deformations of Solid Geometric Models*, Computer Graphics, v20, n4, p151–160, 1986.

[6] Schlick, C., *Wave generators for computer graphics*, Graphics Gems V, Academic Press, p367-374, 1995.

[7] Schlick, C., *A toolbox of macro functions for computer graphics*, Graphics Gems V, Academic Press, 1995.