# *3dLib overview*

3dLIB is a library of pascal units that allows Turbo-Pascal programmers to write applications that display and animate 3D wire-mesh objects. The library is based on a project developed since 1984 on different platforms.

This package supports both Turbo-Pascal for DOS and Turbo Pascal for Windows.

A package called 3D120 is distributed by ISoft D&M that includes a graphic editor, macro interpreter and pascal translator to create and use 3D objects. It is highly recommended that any programmer trying to use this library will use the 3D program to create objects, and understand the animation abilities of the library.

The 3dCW program is a MDI IDE for 3d macro programs (.M3D files).

# *Terminology*

The following terms are described in order to understand the need the building blocks of the 3d environment :

Pascal 3D Types
- Base Object
- Simple Object
- Complex Object

Projections
CTM

# *Pascal 3D Types*

The 3d library defines pascal objects that represent the 3d objects we use to animate. Each object can display itself on the screen, move (translate), rotate and scale itself.

The code for the pascal objects is in the rtobj.pas source code that is supplied with this library.

There are 3 types of pascal objects used to describe the 3d objects :

# *Base Object*

baseObject - defined in the RTOBJ.PAS file, this is the "dummy" ancestor object class of the 3D-objects, such an object has a CTM (called myCTM), a color, a center of gravity 3D point (location), and other display attributes, in order to achieve better animation results, a scrPntUpdt boolean variable is used to indicate if the screen 2D points of the object has to be re-calculated from the 3D representation.

The object uses a 4X4 "transformation matrix" to represent itself in the 3D universe, the move, translate, scale, allScale, rotate, goto3dPos, setToOrigin, calcLocation and deleteTransform methods are used to update that matrix (the Current Transformation Matrix).

The load, save, writeMe, and readMe methods are used to store and retrieve a 3D object from a storage device (a disk..), and the open and close methods are used to construct, and destruct the object. The show, hide and paint methods display, or erase the object on the 2D screen. One more interesting method is the updateScreenPoints method, that transforms the 3D object representation to the 2D screen.

The object's type definition is :

```
    BaseObject = object
        MyCtm          : Ctm;        { This CTM applied to the object gives the   }
                                     {   objects Position after transformations    }
        Name           : String;     { Identifies the object                       }
        myColor        : word;       { Main color for the object                   }
        Location       : point3d;    { Central of gravity in real space            }
        scrPntUpdt     : boolean;    { True if screen points updated               }

        constructor open(myName : string; color : word);
        destructor   CloseMe; virtual;
{$ifdef windows}
        procedure    show(dc : hdc); virtual;
        procedure    hide(dc : hdc); virtual;
        procedure    paint(dc : hdc); virtual; {in specified color}
{$else}
        procedure    show; virtual;
        procedure    hide; virtual;
        procedure    paint; virtual; {in specified color}
{$endif}
        procedure    updateScreenPoints; virtual; {transform object 3D -> 2D}
        procedure    move(axis : axisType; by : real); virtual;
        procedure    translate(dx, dy, dz : integer); virtual;
                  {multy dimentional move in 1 call}
        procedure    scale(axis : axisType; factor : real); virtual;
        procedure    allScale(sx, sy, sz : real); virtual;
                  {multy dimentional scale in 1 call}
        procedure    rotate(axis : axisType; deg : real); virtual;

        procedure    goto3dPos(x, y, z : real); virtual; {translate to absolute place}
        procedure    setToOrigin; virtual;
                  {translate to 0,0,0, update points, and set myCtm to unit}
```

```
        procedure     calcLocation; virtual; {set Location to central gravity}
        procedure     deleteTransform; virtual; {set MyCtm to unit}

        function load : word; virtual; {from disk}
        function save : word; virtual; {to     disk}
        procedure writeMe(var elementFile : f_real); virtual; {to disk .. without opening file..}
        procedure readMe(var elementFile : f_real); virtual;
    end;
```

Related Topics :

Simple Object
Complex Object

# *Simple Object*

A simple object is a descendent of a base object that has the pascal name obj3D.

obj3D is a descendant object of the baseObject class, this is a simple wire-mesh object, that is built from an array of maxPoints points (change this constant in the HDR3D.PAS file to create bigger, or smaller objects) in the 3D universe, an array of maxLines lines (a line is a segment in the 3D universe that connects 2 3D points), another array holds the 2D screen points of the object, calculated from its 3D representation, and the CTM. Another interesting aspect of this object is the use of 2 more matrices, the reverseRot, and unReverseRot CTM objects, that are used to hold only the reverse, and counter reverse of the rotation transformations, these are used by complex objects that contain several obj3D objects, where some of them has to be rotated, scaled and translated around an arbitrary point in the 3D universe, which is not there center of gravity (frame - reference).

The object's type definition is :

```
    Obj3d = object(BaseObject)
        Points        : array[1..MaxPoints] of point3d;
        Lines         : array[1..MaxLines]    of Line3d;
        scrPoints     : array[1..MaxPoints] of screenPoints;
        NumOfLines    : integer;
        NumOfPoints : integer;
        ReverseRot    : Ctm;   { Saves only the reverse rotations }
        unReverseRot: Ctm;   { reverse of the above}

        constructor open(myName : string; ref : point3d; color : word);
        destructor   CloseMe; virtual;
{$ifdef windows}
        procedure    paint(dc : hdc); virtual; {in specified color}
{$else}
        procedure    paint; virtual; {in specified color}
{$endif}
        procedure    updateScreenPoints; virtual; {transform object 3D -> 2D}

        procedure    calcLocation; virtual; {set Location to central gravity}
        procedure    setToOrigin; virtual;

        procedure writeMe(var elementFile : f_real); virtual;
        procedure readMe(var elementFile : f_real); virtual;
    end;
```

Related Topics :

Base Object

Complex Object

# *Complex Object*

The RTOBJ.PAS file contains the definition of the complexObj 3D object, this is an object that contains a maxSubObjects (defined in that file) array of simple wire-mesh obj3D objects. This object class sometimes referred to as the "super - object", allows the user to create complex 3D objects that has a common frame - reference (center of gravity), an example of such an object might be a Robot, that has a center of gravity, and is built of some sub objects that must be able to be transformed both with the frame - reference, and by themselves.

The objects type definition is :

```
    ComplexObj = object(BaseObject)
        childs         : array [1..maxSubObjects] of obj3dPtr;
        ctms           : array [1..maxSubObjects] of ctm;
        numOfChilds : integer; {counter of # of obj3d childs}

        constructor open(myName : string; color : word);
        destructor   closeMe; virtual;
        procedure    updateScreenPoints; virtual;
        procedure    writeMe(var elementFile : f_real); virtual;
        procedure    readMe(var elementFile : f_real); virtual;
        procedure    calcLocation; virtual;
{$ifdef WINDOWS}
        procedure    paint(dc : hdc); virtual;
{$else}
        procedure    paint; virtual;
{$endif}
        procedure    move(axis : axisType; by : real); virtual;
        procedure    rotate(axis : axisType; deg : real); virtual;
        procedure    scale(axis : axisType; factor : real); virtual;

        function     addSubObject(myName : string; refPoint : point3d) : word;
        function     getChildPtr(index : integer) : obj3dPtr;
        procedure    rotateChild(child : integer; axis : axisType;
                                                  deg : real);
        procedure    scaleChild(child : integer; axis : axisType;
                                                  factor : real);
        procedure    moveChild(child : integer; axis : axisType;
                                        by : real);
    end;
```

Related topics :

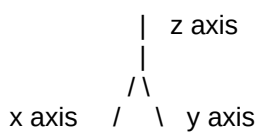
Base Object
Simple Object

# *Projections*

The PROJECT3.PAS file contain the code that transforms objects and points from the 3D universe, to the 2D coordinates.

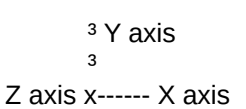
Two 3D -> 2D transformations are supported, axonometric projection, and perspective projection. This is a short explanation of the difference between these 2 projections :

A : axonometric projections, no perspective due to distance is performed, the general way we can look at the coordinate system is as follows :




B : perspective projections : the normal eye perspective projection is performed, we can look at the 3d universe we are referring to as a cube of 1000 x 1000 x 1000 integer locations, with the x axis, and y axis parallel to the screen x, y axis respectively, and the z axis going into the screen.

we will look at the coordinate system as follows :




These units contain a calcPoint procedure that receives a 3D point, and transforms it to a 2D screen coordinate, the setPerspective, resetPerspective and togglePerspective change from perspective projection to axonometric projection, and vice versa.

Related topics :
Terminology
overview

# *Current Transformation Matrix (CTM)*

CTM3D.PAS is the file that defines the current transformation matrix that is used to position the 3D objects in the universe. The CTM is a 4x4 matrix that is multiplied (from the right) by each point of the 3D object whenever a new location is desired for the object.

This unit defines all the transformations that can be applied and used by a CTM, like rotate, scale, translate etc..

Homogeneous Coordinates
---

Homogeneous coordinates allow transformations to be represented by matrices. A 3x3 matrix is used for 2D transformations, and a 4x4 matrix for 3D transformations.

THIS MODULE IMPLEMENTS ONLY 3D TRANSFORMATIONS.

in homogeneous coordination the point P(x,y,z) is represented as P(w*x, w*y, w*z, w) for any scale factor w!=0.
in this module w == 1.

Transformations:

1. translation

[x, y, z] --> [x + Dx, y + Dy, z + Dz]

$$T(Dx, Dy, Dz) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Dx & Dy & Dz & 1 \end{bmatrix}$$

2. scaling

[x, y, z] --> [Sx * x, Sy * y, Sz * z]

$$S(Sx, Sy) = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. rotation

a) Around the Z axis:

[x, y, z] --> [x*cost - t*sint, x*sint + y*cost, z]

$$Rz(t) = \begin{bmatrix} cost & sint & 0 & 0 \\ -sint & cost & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

b) Around the X axis:

[x, y, z] --> [x, y*cost - z*sint, y*sint + z*cost]

$$Rx(t) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos t & \sin t & 0 \\ 0 & -\sin t & \cos t & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

c) Around the Y axis:

[x, y, z] --> [xcost + z*sint, y, z*cost - x*sint]

$$Ry(t) = \begin{bmatrix} \cos t & 0 & -\sin t & 0 \\ 0 & 1 & 0 & 0 \\ \sin t & 0 & \cos t & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

transformation of the vector [x,y,z,1] by transformation matrix T is given by the formula:

$$[x', y', z', 1] = [x,y,z,1] \begin{bmatrix} & & & \\ & T & & \\ & & & \\ & & & \end{bmatrix}$$

Optimizations:

The most general composition of R, S and T operations will produce a matrix of the form:

$$\begin{bmatrix} r11 & r12 & r13 & 0 \\ r21 & r22 & r23 & 0 \\ r31 & r32 & r33 & 0 \\ tx & ty & tz & 1 \end{bmatrix}$$

The task of matrix multiplication can be simplified by

x' = x*r11 + y*r21 + z*r31 + tx
y' = x*r12 + y*r22 + z*r32 + ty
z' = x*r13 + y*r23 + z*r33 + tz

See also:

"Fundamentals of Interactive Computer Graphics" J.D FOLEY A.VAN DAM Adison-Weslely ISBN 0-201-14468-9 pp 245-265

Related Topics :
Terminology

# *Contact*

Please contact :

ISoft D&M,
P.O.B 5517
Coralville IA 52241,
U.S.A

To contact the author directly :

Contact : Loewy Ron,
9 Haneveem st.
Herzeliya, 46465
ISRAEL.

# *Registration*

3dLIB is a shareware product, if you find this product valuable, please register it. This section describes the reasons you should register.

By registering you will receive a diskette with the latest 3dLIB version, the 3d environment program, for WYSIWYG object creation, the complete source code for the 3d environment program, and - you will help us to create the next version of 3dLIB - that will include even more features then the features that are currently available!, we might even add YOUR enhancement requests!

# *Macro Language (M3D)*

The following commands are supported in the 3D Macro language :

c - Clear Screen .
w - set Color to White.
b - set Color to Black.
p - Paint Active object in last color.
o0 - set Perspective off.
o1 - set Perspective on.
>x - Start a loop to be performed x times.
lfl - Load simple object from file fl into the active object.
Lfl - Load complex object from file fl into the active object.
rad - Rotate Active Element In axis a, d degrees.
ex - Choose Active Element x.
mas - Move in a axis, s steps.
gx,y,z - Goto 3d pos x,y,z.
saf - Scale a axis in f factor.
v0..v9 variable names.
< - End Loop.
\ - End line.
z - set object to center.
Rsad - rotate a sub object of a complex object
around axis a, d degrees.
Msas - move a sub object of a complex object
in axis a, s steps.
Ssaf - scale a sub object of a complex object
around axis a, by a factor of f.

Special Notes : a number must end with a space character.
Loops can be nested 10 levels deep in the 3D111 package,
but 3DC can translate even more levels that.

any numeric expressions requested can be
given with a normal infix notation, for
example :

rx30 * v1 + 2

will rotate the current active object
around the x axis, by 2 plus 30 * v1
degrees, where v1 is a variable.

# *CREDITS*

3dLIB was written using Turbo Pascal 6.0 & 7.0, as well as
Turbo Pascal for Windows 1.0, 1.5 and Borland Pascal with objects 7.0.
These products are trademarks of Borland international.

Windows 3.0 and Windows 3.1 are trademarks of Microsoft Corp.

The Borland Brief v3.1 editor was used to write these programs.

The windows help file was created using the help development kit v2.0
by Loewy Ron.

The Author's picture (in the 3dCW program) was taken by Allison Bially.

Related Topics : Overview