eyes

Overview

Bsp is a user interface developed for the *Pixel-planes* graphics engine. Its name stands for Binary Space Partition tree, a polygon sorting algorithm developed by Henry Fuchs and Greg Abram. Bsp uses this algorithm to render an object with hidden surface removal without the use of a z-buffer. Bsp expects as input a file containing an object sorted into a bsp-tree. After this file is read, the user manipulates the position and orientation of his viewpoint, and the object is rendered by traversing the bsp-tree to plot the polygons of the object from back to front, implementing hidden surface removal. Along with the resulting image on *Pixel-planes*, the user is presented with a display on the host computer. This display includes three orthogonal projections of object space, giving the user his position and orientation, and a crude copy of the image on *Pixel-planes*, useful in debugging.

Input

Bsp is called as follows: bsp [[-]b] [[-]n] [[-]i] [[-]d] inputfile, where the options are

(-)b : backfacing colors (not) present in inputfile
(-)n : vertex normals (not) present in inputfile
(-)i : (don't) send commands to the igc
(-)d : (don't) print the igc commands on stdout

The default options are b, n, i, and -d.

Bsp expects inputfile to be the output of the program TConvert in Greg Abram's BSP-tree package, which can be found in the directory grumpy:~gda/release. Bsp normally expects both vertex normals and backfacing polygon colors in its input. These options can be turned off with the command line flags -n and -b, respectively.

The BSP-tree package is documented in the file grumpy:~gda/release/ReadMe. Probably the only command you will ever need from this package is the pipe

PConvert -b < file.poly | Vnorm -b | Treemaker -b -n | TConvert -b -n > file.tree,

where file.poly is essentially a list of polygons, and file.tree is the output suitible for input to Bsp. Several further notes:

1. Bsp uses a left-handed coordinate system. In the input to the BSP-tree package, polygon vertices should be listed clockwise, looking at the front face of the polygon.

2. Some object creation systems which generate input compatible with the BSP-tree package allow polygon colors to be specified as -1, meaning to copy the color of the previous polygon. Unfortunately, these -1's simply propagate through the BSP-package and generate incorrect results, as the polygon order is completely scrambled by the treemaker.

3. The progam Convert in the BSP-tree package is not needed, as Bsp automatically centers and normalizes its input.

Coordinate Spaces

The object passes through several coordinate spaces on its way to the masscomp and *Pixel-planes* screens. *Object space* is the space in which the object is originally defined. The viewer's position is defined by the

location of the ***eyepoint*** in object space, and his orientation is defined by the orientation of the ***eyespace cartesian vectors*** in object space.

An eyepoint and its cartesian vectors are used to create a transformation matrix which transforms the object from object space to ***eye space,*** where the viewer is at the origin looking down the z-axis. The x-axis is to his right and the y-axis is up. Hither clipping is performed in screen space. The light's position is defined in eye space.

There are two ***screen spaces*** in the system: one for the masscomp screen and one for *Pixel-planes.* The object is transformed into these two spaces from eye space by first dividing the x and y coordinates of each vertex by its z coordinate, to give perspective. The x and y coordinates are then scaled and shifted appropriately in each screen space to place the image in the correct place on the screen, with the correct aspect ratio.

Object space, eye space, and the masscomp screen space are left handed coordinate spaces. For some reason, the hardware geeks placed the origin of *Pixel-planes* (otherwise an excellent system) in the ***upper*** left corner, causing its screen space to be right handed.

Main Routines

Each paragraph in this section describes either a single major routine or a file containing a collection of related routines. The sources for the major routines are in files of the same name with a '.c' suffix.

Main parses the command line, opens the input file, and starts up the system.

InitGP, called by main, sets up the graphics windows on the masscomp and loads the masscomp color map. Also turns off the text cursor.

DropGP redirects text back to the full masscomp screen, turns on the text cursor, and gives up the graphics processor. This routine should be called before exiting at any point after the InitGP call.

GetTree, called by main, reads the inputfile to determine the number of polygons and vertices are in the object, then reads the file into the object space data structures.

Allocate is called by GetTree to allocate storage for the object data structures, once GetTree has determined the number of polygons and vertices.

ReadNode is called by GetTree to read one polygon (a node in the bsp-tree) from inputfile.

Normalize is called by main to center and normalize the object.

MakeObjSeg, called by main, creates the display segment which draws the screen space polygons in the object window on the masscomp screen. This display segment is submitted to the graphics processor by Refresh after each traversal of the bsp-tree.

MakeProjSeg, called by main, creates the display segment which draws the three orthonormal projections of object space in the three projection windows on the masscomp screen. Project submits this segment whenever the scale in the projection windows needs to change.

MakeFrustSeg, called by main, creates the display segment which draws the view frustrums and light dots in the projection windows on the masscomp screen. Project submits this segment each time it is called.

CmdLoop, called by main, is a small piece of initialization code followed by a loop containing a giant switch statement which calls other routines based upon which key the user is holding down. The loop ends when the user hits the quit key.

Poll.c contains a collection of routines which are called by CmdLoop to handle the masscomp keyboard.

InitView is called by CmdLoop to initialize the position and orientation of the eyepoint, the position of the light, and other parameters defining the view.

ManipView.c contains a collection of routines which are called by CmdLoop to change the position and orientation of the eyepoint, and the angle of view.

MoveLight is called by CmdLoop to move the light around in eyespace.

PromptView is called by CmdLoop to prompt the user to type in several view-related parameters.

Help is called by CmdLoop to print the contents of the help file.

Refresh is called by CmdLoop to update the eyespace and screenspace data structures, and refresh the masscomp and *Pixel-planes* screens.

MakeXforms is called by Refresh to create the transformation matrix for transforming from object space to eye space, based on the position and orientation of the eyepoint. MakeXforms also creates other matrices used by Project in drawing the projections of the viewing frustum and light position.

Project is called by Refresh to update the three projection windows on the masscomp screen.

Transform is called by Refresh to transform the object from the object space data structures into the eye space data structures, and to calculate the polygon and vertex colors based on the position of the light.

HitherClip, called by Refresh, performs three functions: it clips the object in eye space to the hither plane, ensures that the vertices are listed counter-clockwise looking down the eye space z-axis, and transforms the object vertices into the masscomp screenspace data structure.

BspTrav, called by Refresh, traverses the bsp-tree in the order specified by the position of the eyepoint, calling Write at each node.

Write is called by BspTrav at each node (polygon) of the bsp-tree to output that polygon to the *Pixel-planes* translator and to place that polygon in the data structure used by the masscomp graphics processor to draw the scene.

ScreenOps.c contains a collection of routines which are called by CmdLoop to perform simple updates on the masscomp screen.

PointOps.c contains a collection of routines for performing vector additions, dot products, etc.

MatrixOps.c contains a collection of routines for creating and manipulating matrices.

Translator.c contains three routines which make up the software translator, to be replaced with Justin Heinecke's translator board. *InitTrans* reads files in /usr/pxpl to initialize constants and to load the igc microcode instruction and parameter words. *NewFrame,* called by Refresh, initializes constants used by Poly to transform to *Pixel-planes* screenspace, and generates the igc commands to clear the *Pixel-planes* screen. *Poly,* called by Write, generates the igc commands to draw a single smooth-shaded polygon on *Pixel-planes.*

ToIgc.c contains the debug versions of the routines IgcWrite and IgcStream, which are called by the routines in Translator.c to output the igc commands to stdout when the debug (d) option has been selected in the command line.