developers

COLLABORATORS			
	TITLE : developers		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 24, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

developers

1.1 developers.guide

```
                INFORMATION FOR DEVELOPERS


                Real 3D Binary File Format

                Display Driver Interface
```

1.2 binary format

```
                REAL 3D BINARY FILE FORMAT DESCRIPTION


This information is provided for developers who want to write programs
capable of reading and writing Real 3D IFF files.


The format uses IEEE double precision to store floating point values.
Natural, recommended magnitude of spatial coordinates is between 0
and 2. The algorithms have been designed so that the best accuracy is
obtained when coordinates range within this interval.

Coordinates are expressed using the following structure:

    typedef struct vector
    {    double r, s, t;
    } VECTOR;

The coordinate system orientation is such, that when looking at the object
in a front view, x-axis (r-component) points to the right, y-axis
(s-component) points upwards, and z-axis (t-component) points towards the
camera.


The name of the IFF FORM is REAL.
```

Real 3D IFF file consists of the following chunks:

```
        Chunkc ID Explanation
        ----------------------------------------

        ROBJ
                Object Data
                        RMTR
                Material Data
                        RVRS
                Version Data
                        RWIN     Window Description
        RATT     Default Primitive Attributes
        RSTT     Global Settings
        RSCR     Screens
        RCOL     Predefined Colors
        RINF     Measuring System
        RREN     Rendering Settings
        RANI     Animation Settings
        RGRI     Grid Settings
        RRPL     RPL Ascii File
        RCMD     Command Hunk
```

1.3 robj

OBJECT DATA

The ROBJ chunk defines the geometrical and hierarchical structure
of Real 3D objects.

The internal structure of the chunk is the following:

OBJECT NODE

Structure used for storing one hierarchy node to a file.
This is needed for each hierarchy node regardless of its type
(primitive, level, etc).

The structure is the following:

```
struct r3object
{   char  name[16];         /* Object name */
    UBYTE link;             /* See link flags below */
    UWORD type;             /* See Object Types */
    ULONG flags;            /* See Object Flags */
    LONG  reserved[2];      /* Set to 0L */
};
```

Link Flags define the hierarchical structure of the object. When

the ‘r3object‘ structure is read, the reader can find out what comes next by studying these bits. The object data forms a binary tree: each node may have ’next’ and ’sub’ (= child) nodes. Also some other data can be linked to an object node. The existence of such data can be seen from the link flags.

```
#define R3OB_NEXT   (1 << 0)
#define R3OB_SUB    (1 << 1)
#define R3OB_DATA   (1 << 2)
#define R3OB_TAGS   (1 << 3)
#define R3OB_WIRE   (1 << 4)
```

For example, if you want to write an object with a sub-object, set R3OB_SUB flag when writing r3object to the file, and after that write the sub-object.

Object Type defines the type of the node. It can be one of the following:

```
#define CSG_AND             1   /* Boolean AND  node */
#define CSG_OR              2   /* Normal level node */
#define PRIM_RECTANGLE  10
#define PRIM_CUBE           11
#define PRIM_PYRAMID        12
#define PRIM_CUTPYRAMID     13
#define PRIM_POLYGON        14
#define PRIM_POLYHEDRON     15
#define PRIM_POLYMID        16
#define PRIM_CUTPOLYMID     17
#define PRIM_ELLIPSE        18
#define PRIM_CYLINDER       19
#define PRIM_CONE           20
#define PRIM_CUTCONE        21
#define PRIM_ELLIPSOID      22
#define PRIM_HYPERBOL       23
#define PRIM_ELLIPSEG       24
#define PRIM_SPLINE         26
#define PRIM_GROUP          27
#define PRIM_SYMLINK        28
#define PRIM_TRISET         29
#define PRIM_OFFSET         33
#define PRIM_LINE           34
#define PRIM_COORD          35
#define PRIM_OBSERVER       36 /* Viewpoint */
#define PRIM_AIMPOINT       37
#define PRIM_ATTR           38
```

Object Flags field defines On/Off kind of object attributes. Bits in this field correspond the contents of the Modify/Properties/Attributes requester.

```
#define OBJ_INVERTED    (1L<<0)  /* The volume is inverted */
#define OBJ_PAINTER     (1L<<1)  /* Boolean with material */
#define OBJ_INVISIBLE   (1L<<2)  /* Invisible in editor */
#define OBJ_LIGHTSOURCE (1L<<3)  /* lightsource */
```

```
#define OBJ_HOLLOW      (1L<<4)  /* Surface instead of Solid */
#define OBJ_INFINITE    (1L<<5)
#define OBJ_SCENE       (1L<<6)  /* Invisible in first stage RT */
#define OBJ_CONTROL     (1L<<7)  /* Control element, invisible in RT  ↩
   */
#define OBJ_PROTECTL2   (1L<<8)  /* Protected against move, color etc. ↩
    */
#define OBJ_UNCOVERED1  (1L<<9)  /* No bottom */
#define OBJ_UNCOVERED2  (1L<<10) /* No cover */
#define OBJ_TEXTURE     (1L<<11) /* Texture mapping description */
#define OBJ_SECTOR      (1L<<12) /* Sector primitive */
#define OBJ_PROTECTED   (1L<<13) /* protected against deletion etc. */
#define OBJ_NOSECTSURF  (1L<<14) /* Only quadr. surface is rendered */
#define OBJ_NOREFL      (1L<<15) /* Not reflected by other objects */
#define OBJ_RESERVED1   (1L<<16) /* set to 0 */
#define OBJ_MOTION      (1L<<17) /* motion blurred object */
#define OBJ_SHADOWLESS  (1L<<18) /* does not cast shadows */
#define OBJ_MATTE       (1L<<19) /* background colored object */
#define OBJ_USERREPR    (1L<<28) /* user defined wireframe */
```

All other bits are used internally and MUST be set to zero.

PRIMITIVE

If the object is primitive, there is a primitive data associated with it.

Structure used for saving primitive data is described below and is always written next to r3object.

```
struct r3primitive
{   UBYTE reg;              /* Color register for wireframe */
    COLOR color;            /* RGBA-color */
    ULONG expansion;        /* set to 0L */
};
```

The COLOR structure is defined below

```
typedef struct r3color
{   UBYTE R; /* 0 - 255 */
    UBYTE G;
    UBYTE B;
    UBYTE A; /* Alpha channel value */
}COLOR;
```

GEOMETRY

If the object is 'visible', there is a geom. description associated with the primitive data. Size and structure of this structure depend on the type of the primitive in question.

See primitive data structures below.

CUSTOM REPRESENTATION

If the wire frame representation of object cannot be recreated
from the geometry, it is written after the geometry. This is
true if the OBJ_USERREPR flag is set in r3object structure.

The header structure of custom representation is as follows:

```
struct r3wire
{   UWORD point_num;    /* number of 3D points */
    UWORD join_num;     /* size of join array */
    UWORD pattern;      /* line type */
    ULONG expansion;    /* set to 0L */
};
```

Then comes the actual data in the following order:

```
VECTOR points[point_num];
WORD  join[join_num];
```

The join table contains index numbers referring to the
wireframe points; index '1' refers to points[0] and so on.
Zero acts as a separator (pen up).

GEOM. EXPANSION

Depending on the type of the object, there can be extra data
associated with it. The size and the structure of this data
can be detected from the type of the object.

TAGS

If there are tags linked to r3object, they are written last.

```
/*
*  Real 3D control tag identifiers.
*/

#define RTAG_END        0L

/*
 * Tag data types. First byte of Tag_ide is one of these,
 * unless Tag_ide is a control identifier.
 */

#define RTAG_FLT     'F' /* floating point value */
#define RTAG_INT     'I' /* long integer value */
#define RTAG_STR     'S' /* character string */
#define RTAG_VEC     'V' /* general vector */
#define RTAG_MVE     'M' /* these tags are modified as points */
#define RTAG_CTR     'C'
#define RTAG_DVE     'D' /* these tags are modified as vectors */

#define RTAG_TYPECHAR(tagid) ((LONG)(tagid)>>24L)

typedef struct rtag
{   long tag_ide;  /* first byte one of RTAG_xxx defs above */
```

```
          void *tag_val;
        } RTAG;



PRIMITIVE GEOMETRY

        /*
         *  Structures used for defining primitives.
         */

        typedef struct polygon  /* polygon, polyhedron, polymid */
        {   UWORD count;        /* number of points in primitive */
            VECTOR *points;     /* array of points */
            VECTOR stick;       /* thickness of polyhedron or top of polymid */
        } POLYGON;              /* or boolean volume direction of polygon */

  typedef struct cutpolymid
  {   UWORD count;    /* number of points in the base */
      VECTOR *points; /* 2*count points */
  } CUTPOLYMID;

  typedef struct rectangle  /*rectangle, cube, pyramid */
  {   VECTOR points[3]; /* Three points defining rectangular base */
      VECTOR stick;     /* Top of pyramid or thickness of cube */
  } RECTANGLE;          /* or boolean volume indicator of rectangle */

  typedef struct cutpyramid  /* Three points defining rectangular base */
  {   VECTOR points[6];      /* + three points defining rectangular top */
  } CUTPYRAMID;

     /* In the following primitives, bounding planes are defined by
        a point 'p' in the plane and two vectors 'n' and 'm', whose
        cross product is the normal vector of the plane. Bounding planes
        are used to delimit suitable parts of different primitives. */

        typedef struct quadric  /* cylinder, cutcone, hyperboloid, ellipseg */
  {   VECTOR center, a, b, c;         /* center and three axis */
      VECTOR p1, n1, m1, p2, n2, m2;  /* Two bounding planes */
      float as, ae;   /* start/end angles for sectors */
  } QUADRIC;          /* used when OBJ_SECTOR object flag is set */

  typedef struct cone
  {   VECTOR center, a, b, c; /* center = top peak, c = cone axis */
      VECTOR p, n, m;         /* bounding plane */
      float as, ae;           /* start/end angles */
  } CONE;

  typedef struct ellipsoid
  {   VECTOR center;
      VECTOR a, b, c; /* axes */
      float as, ae;   /* start/end angles */
  } ELLIPSOID;

  typedef struct spline
  {   UWORD width, height;
      VECTOR *matrix; /* width * height points defining control hull */
```

```
    UWORD flags;    /* see freeform flags below */
    WORD type;      /* see freeform types below */
} SPLINE;

typedef struct ellipse
{   VECTOR center, a, b;
    VECTOR stick;
    UWORD flags;    /* see CTRL_ flags above */
    float as, ae;   /* start/end angles (defined in object space) */
} ELLIPSE;

typedef struct line
{   UWORD count;
    VECTOR *points;
    WORD type;      /* see freeform types below */
    UWORD flags;    /* see freeform flags below */
} LINE;

typedef struct apoint
{   VECTOR position;  /* Just a single 3D point */
} APOINT;

typedef struct viewpoint
{   VECTOR points[3];        /* left and right eye, direction */
    float dof, dofscale;     /* depth of field stuff, 0 - 3 */
    float scale;             /* zoom scale = camera lens angle */
} VIEWPOINT;                 /* default zoom scale = 1.0 */

typedef struct coordsys
{   VECTOR position, x, y, z; /* center and directions */
} COORDSYS;

typedef struct subprim /* this primitive uses tag for describing parent name */
{   UWORD count;   /* Number of points in this subgroup */
    int *points;   /* Set to 0L */
    OBJECT *owner; /* Set to 0L */
} SUBPRIM;


typedef struct face     /* Used by triset primitive below */
{   UWORD vertexind[3];
    COLOR color;        /* Unused at the moment */
} FACE;

      typedef struct triset
      {   UWORD count;    /* number of points in triset */
          ULONG reserv1;  /* set this to 0L */
          UWORD facenum;  /* number of faces */
          ULONG reserv2;  /* set this to 0L */
          WORD  type;     /* freeform type FT_POLYGON or FT_PHONG */
      } TRISET;

/*
 *  Types. Evaluation for control mesh/curve can be:
 */

#define FT_POLYGON      0  /* Just line/polygonal surface */
```

```
  #define FT_PHONG        1  /* polygonal based phong shaded surface */
  #define FT_BSPLINE      3  /* cubic B-spline */

  /*
   *  Freeform flags
   */

  #define FF_WRAP_U (1<<0)  /* close surface in `u` (width) direction */
  #define FF_WRAP_V (1<<1)  /* close `v` (height) direction */
```

CONVERTING POLYGONAL OBJECTS

The best way to represent polygonal freeform objects in Real 3D is to use `triset` primitive which consists of a number of vertices and faces.

B-Spline meshes are the most natural choice from Real 3D`s stand point, but this usually requires that the object is originally modelled using rectangular meshes.

The following diagram shows the contents of a file consisting of one triset primitive.

```
    FORMxxxxREAL

    RVRSxxxx
        struct r3version

    ROBJxxxx
        struct r3object
        struct r3primitive
        struct triset
        VECTOR points[count];   /* `triset->count number` of points first */
        FACE faces[facenum];    /* `triset->facenum number` of faces next */
```

COMPLETE READING EXAMPLE

```
/*
 *  Memory resident structures. The link bits are replaced with pointers,
 *  so that the data can be linked hierarchically in memory.
 */

typedef struct object
{   struct object *next;
    struct object *sub;     /* descendants */
    UWORD type;             /* CLASS ID see primitives & CSG types above */
    char name[16];
    ULONG flags;            /* see object flags above */
    struct primitive *data; /* if primitive, points to primitive data */
    struct rtag *tags;      /* extensions */
} OBJECT;

typedef struct primitive
```

```
{   void *descr;            /* primitive specific geom. description */
    COLOR color;            /* accurate RGB-color for object */
    UBYTE reg;
} PRIMITIVE;


/*
 *  Some convenient definitions
 */

#define DESCPTR(o)  ((o)->data->descr)
#define PPTR(obj)  (((POLYGON *)(DESCPTR(obj)))->points)
#define PCNT(obj)  (((POLYGON *)(DESCPTR(obj)))->count)
#define SPLPTR(obj) (((SPLINE *)(DESCPTR(obj)))->matrix)
#define SPLCNT(obj) (((SPLINE *)((int)DESCPTR(obj)))->width*(int)((SPLINE *)( ↩
   DESCPTR(obj)))->height)
#define LINEPTR(obj) (((LINE *)(DESCPTR(obj)))->points)
#define LINECNT(obj) (((LINE *)(DESCPTR(obj)))->count)



        int ObjReadCSGNode(OBJECT *obj, FILE *fh)
        {
            struct r3object r3ob;

            if(fread((char *)&r3ob, sizeof(r3ob), 1, fh) != 1)
                return(FALSE);

            obj->type = r3ob.type;
            obj->flags = r3ob.flags;
            strcpy(obj->name, r3ob.name);

            obj->next = (OBJECT *)    (r3ob.link & R3OB_NEXT);
            obj->sub  = (OBJECT *)    (r3ob.link & R3OB_SUB);
            obj->data = (PRIMITIVE *) (r3ob.link & R3OB_DATA);
            obj->tags = (RTAG *)      (r3ob.link & R3OB_TAGS);

            return(TRUE);
        }


        int ObjReadWire(OBJECT *obj, FILE *fh)
        {
            struct r3wire wire;
            VECTOR *points=NULL;
            WORD *joints=NULL;

            if(!fread((char *)&wire, sizeof(wire), 1, fh))
                return(FALSE);

            if(!ObjReadPrimData(&points, wire.point_num, fh))
                goto error;

            if(!(joints = AllocMem(wire.join_num * sizeof(WORD), 0)))
                goto error;
```

```
        if(fread((char *)joints, sizeof(WORD), wire.join_num, fh)!= wire. ↩
            join_num)
            goto error;

        /* Now, what to do with the wireframe data ? */
    /* Let's just discard it */
        FreeMem(points, sizeof(VECTOR) * wire.point_num);
        FreeMem(joints, sizeof(WORD) * wire.join_num);
        return(TRUE);

    error:
        if(points) FreeMem(points, sizeof(VECTOR) * wire.point_num);
        if(joints) FreeMem(joints, sizeof(WORD) * wire.join_num);

        return(FALSE);
    }


    int ObjReadPrimNode(OBJECT *obj, int descsize, FILE *fh)
    {   int answ = FALSE;

        struct r3primitive hdr;
        PRIMITIVE *prim = obj->data;

        if(!(prim->descr = AllocMem(descsize, MEMF_CLEAR)))
            return(FALSE);

        if(fread((char *)&hdr, sizeof(hdr), 1, fh) != 1)
            goto cleanexit;

        if(fread((char *)prim->descr, descsize, 1, fh) != 1)
            goto cleanexit;

        prim->color = hdr.color;
        prim->reg = hdr.reg;

        if(obj->flags & OBJ_USERREPR)
            if(!ObjReadWire(obj, fh)) goto cleanexit;

        answ = TRUE;

    cleanexit:
        if(!answ) FreeMem(prim->descr,descsize);

        return(answ);
    }


    int ObjReadPrimData(VECTOR **points, int count, FILE *fh)
    {
        if(!(*points = AllocMem(count*VECTOR, MEMF_CLEAR)))
            return(FALSE);

        if(fread((char *)*points, sizeof(VECTOR), count, fh) == count)
            return(TRUE);
    error:
        FreeMem(*points, count*VECTOR); *points = NULL;
```

```
    return(FALSE);
}

int ObjReadNode(OBJECT **obj, FILE *fh)
{
    if(!(*obj = (OBJECT *)AllocMem(sizeof(OBJECT), MEMF_CLEAR)))
        return(FALSE);

    if(!ObjReadCSGNode(*obj, fh))
    {   FreeMem(*obj, sizeof(OBJECT));
        *obj = NULL;
        return(FALSE);
    }

    if((*obj)->data)
    {   if(!((*obj)->data = (PRIMITIVE *)AllocMem(sizeof(PRIMITIVE),  ↩
       MEMF_CLEAR)))
        {   FreeMem(*obj, sizeof(OBJECT));
            *obj = NULL;
            return(FALSE);
        }
    }

    switch((*obj)->type)
    {
        case CSG_AND:
        case CSG_OR:
      return(TRUE);

        case PRIM_SYMLINK:
  case PRIM_ATTR:
            return(TRUE);

        case PRIM_RECTANGLE:
        case PRIM_CUBE:
        case PRIM_PYRAMID:
            if(!ObjReadPrimNode(*obj, sizeof(RECTANGLE),  fh))
                goto error;
            break;

        case PRIM_CUTPYRAMID:
            if(!ObjReadPrimNode(*obj, sizeof(CUTPYRAMID),  fh))
                goto error;
            break;

        case PRIM_POLYGON:
        case PRIM_POLYHEDRON:
        case PRIM_POLYMID:
            if(!ObjReadPrimNode(*obj, sizeof(POLYGON), fh))
                goto error;

            if(!ObjReadPrimData(&PPTR(*obj), PCNT(*obj), fh))
            {   FreeMem(DESCPTR(*obj), sizeof(POLYGON));
                goto error;
            }
            break;
```

```
case PRIM_CUTPOLYMID:
    if(!ObjReadPrimNode(*obj, sizeof(CUTPOLYMID), fh))
        goto error;

    if(!ObjReadPrimData(&PPTR(*obj), 2*PCNT(*obj), fh))
    {   FreeMem(DESCPTR(*obj), sizeof(CUTPOLYMID));
        goto error;
    }
    break;

case PRIM_ELLIPSE:
    if(!ObjReadPrimNode(*obj, sizeof(ELLIPSE), fh))
        goto error;
    break;

case PRIM_CONE:
    if(!ObjReadPrimNode(*obj, sizeof(CONE), fh))
        goto error;
    break;

case PRIM_ELLIPSOID:
    if(!ObjReadPrimNode(*obj, sizeof(ELLIPSOID), fh))
        goto error;
    break;

case PRIM_HYPERBOL:
case PRIM_CUTCONE:
case PRIM_ELLIPSEG:
case PRIM_CYLINDER:
    if(!ObjReadPrimNode(*obj, sizeof(QUADRIC), fh))
        goto error;
    break;

case PRIM_SPLINE:
{   if(!ObjReadPrimNode(*obj, sizeof(SPLINE), fh))
        goto error;
    if(!ObjReadPrimData(&SPLPTR(*obj), SPLCNT(*obj), fh))
    { FreeMem(DESCPTR(*obj), sizeof(SPLINE));
        goto error;
    }
    break;
}

case PRIM_OFFSET:
case PRIM_AIMPOINT:
    if(!ObjReadPrimNode(*obj, sizeof(APOINT), fh))
        goto error;
    break;

case PRIM_OBSERVER:
    if(!ObjReadPrimNode(*obj, sizeof(VIEWPOINT), fh))
        goto error;
    break;

case PRIM_LINE:
    if(!ObjReadPrimNode(*obj, sizeof(LINE), fh))
        goto error;
```

```
                if(!ObjReadPrimData(&LINEPTR(*obj), LINECNT(*obj), fh))
                {   FreeMem(DESCPTR(*obj), sizeof(LINE));
                    goto error;
                }
                break;

            case PRIM_TRISET:
            {   TRISET *ts;

                if(!ObjReadPrimNode(*obj, sizeof(TRISET), fh))
                    goto error;
                ts = DESCPTR(*obj);
                if(!ObjReadPrimData(&PPTR(*obj), PCNT(*obj), fh))
                {   FreeMem(DESCPTR(*obj), sizeof(TRISET));
                    goto error;
                }
                if(!(ts->faces = AllocMem(ts->facenum * sizeof(FACE), 0L)))
                {   FreeMem(PPTR(*obj), sizeof(VECTOR)*PCNT(*obj));
                    FreeMem(DESCPTR(*obj), sizeof(TRISET));
                    goto error;
                }
                if(fread((char *)ts->faces, sizeof(FACE), ts->facenum, fh) <  ↩
                   ts->facenum)
                {   FreeMem(ts->faces, sizeof(FACE) * ts->facenum);
                    FreeMem(PPTR(*obj), sizeof(VECTOR)*PCNT(*obj));
                    FreeMem(DESCPTR(*obj), sizeof(TRISET));
                    RealError(ERR_READ);
        goto error;
                }
                break;
            }

            case PRIM_COORD:
                if(!ObjReadPrimNode(*obj, sizeof(COORDSYS), fh))
                    goto error;
                break;

            case PRIM_GROUP:
                if(!ObjReadPrimNode(*obj, sizeof(SUBPRIM), fh))
                    goto error;
                if(!(ObjReadGroup(*obj, fh)))
                {   FreeMem(DESCPTR(*obj), sizeof(SUBPRIM));
                    goto error;
                }
                break;

            default:
                RealError("UNKNOWN OBJECT TYPE");
                return(FALSE);
        }
        return(TRUE);

    error:
        FreeMem((*obj)->data, sizeof(PRIMITIVE));
        FreeMem(*obj, sizeof(OBJECT));
        *obj = NULL;
        return(FALSE);
```

```
        }

        int ObjReadLevel(OBJECT **obj, FILE *fh)
        {
            for(; *obj; obj = &(*obj)->next)
            {   if(!ObjRead(obj, fh))
                    return(FALSE);
            }
            return(TRUE);
        }

        int ObjRead(OBJECT **obj, FILE *fh)
        {
            if(!ObjReadNode(obj, fh))
                return(FALSE);

            if((*obj)->tags)
                if(TagRead(&(*obj)->tags, fh) == FALSE)
                    return(FALSE);

            if((*obj)->sub)
                if(ObjReadLevel(&(*obj)->sub, fh) == FALSE)
                    return(FALSE);

            return(TRUE);
        }

        int ObjReadGroup(OBJECT *obj, FILE *fh)
  {
      struct subprim *sb = DESCPTR(obj);

      if(sb->count) /*symbolic link or group */
      {   if(!(sb->points = (int *)AllocMem(sb->count*sizeof(int), 0)))
              return(FALSE);
          if(fread((char *)sb->points, sizeof(int), sb->count, fh) != sb->count)
          {   FreeMem(sb->points, sizeof(int) * sb->count);
              RealError(ERR_READ);
              return(FALSE);
          }
      }
      sb->owner = NULL;
      return(TRUE);
  }
```

COMPLETE WRITING EXAMPLE

```
  int ObjSaveCSGNode(FILE *fh, OBJECT *obj)
  {
      struct r3object r3ob;

      r3ob.type   = obj->type;
      r3ob.flags  = obj->flags;
      strcpy(r3ob.name, obj->name);
            r3ob.reserved[0] = r3ob.reserved[1] = 0;
```

```
    if(obj->next) r3ob.link |= R3OB_NEXT;
    if(obj->sub)  r3ob.link |= R3OB_SUB;
    if(obj->data) r3ob.link |= R3OB_DATA;
    if(obj->tags) r3ob.link |= R3OB_TAGS;

    if(fwrite((char *)&r3ob, sizeof(r3obj), 1, fh) != 1)
    {   RealError(ERR_WRITE);
      return(FALSE);
    }
    return(TRUE);
}

int ObjSavePrimNode(FILE *fh, OBJECT *obj, int descsize)
{
    struct r3primitive hdr;
    struct primitive *prim = obj->primitive;

    memset(&hdr, 0, sizeof(hdr));
    hdr.reg   = prim->wire.reg;
    hdr.color = prim->color;

    if(fwrite((char *)&hdr, sizeof(struct r3primitive), 1, fh) != 1)
        goto error;
    if(!fwrite((char *)prim->descr, descsize, 1, fh) == 1)
        goto error;
    return(TRUE);
error:
    RealError(ERR_WRITE);
    return(FALSE);
}

int ObjSavePrimData(FILE *fh, VECTOR *points, int count)
{
    if(fwrite((char *)points, sizeof(VECTOR), count, fh) == count)
        return(TRUE);
    RealError(ERR_WRITE);
    return(FALSE);
}

int ObjSaveGroup(FILE *fh, OBJECT *obj)
{
    struct subgroup *sb = DESCPTR(obj);

    if(!ObjSavePrimNode(fh, obj, sizeof(struct subprim)))
        return(FALSE);
    if(sb->count)
        if(fwrite((char *)sb->points, sizeof(int), sb->count, fh) != sb->count)
            return(FALSE);
    return(TRUE);
}


int ObjSaveNode(FILE *fh, OBJECT *obj)
{
    if(!ObjSaveCSGNode(fh, obj))
        return(FALSE);
```

```
switch(obj->type)
{
    case CSG_AND:
    case CSG_OR:
        return(TRUE);

    case PRIM_SYMLINK:
    case PRIM_ATTR:
  return(TRUE);

    case PRIM_RECTANGLE:
    case PRIM_CUBE:
    case PRIM_PYRAMID:
        return(ObjSavePrimNode(fh, obj, sizeof(RECTANGLE)));

    case PRIM_CUTPYRAMID:
        return(ObjSavePrimNode(fh, obj, sizeof(CUTPYRAMID)));

    case PRIM_POLYGON:
    case PRIM_POLYHEDRON:
    case PRIM_POLYMID:
        if(!ObjSavePrimNode(fh, obj, sizeof(POLYGON)))
            return(FALSE);
        return(ObjSavePrimData(fh, PPTR(obj), PCNT(obj)));

    case PRIM_CUTPOLYMID:
        if(!ObjSavePrimNode(fh, obj, sizeof(CUTPOLYMID)))
            return(FALSE);
        return(ObjSavePrimData(fh, PPTR(obj), 2*PCNT(obj)));

    case PRIM_ELLIPSE:
        return(ObjSavePrimNode(fh, obj, sizeof(ELLIPSE)));

    case PRIM_CONE:
        return(ObjSavePrimNode(fh, obj, sizeof(CONE)));

    case PRIM_ELLIPSOID:
        return(ObjSavePrimNode(fh, obj, sizeof(ELLIPSOID)));

    case PRIM_HYPERBOL:
    case PRIM_CUTCONE:
    case PRIM_ELLIPSEG:
    case PRIM_CYLINDER:
        return(ObjSavePrimNode(fh, obj, sizeof(QUADRIC)));

    case PRIM_SPLINE:
        if(!ObjSavePrimNode(fh, obj, sizeof(SPLINE)))
            return(FALSE);
        return(ObjSavePrimData(fh, SPLPTR(obj), SPLCNT(obj)));

    case PRIM_OFFSET:
    case PRIM_AIMPOINT:
        return(ObjSavePrimNode(fh, obj, sizeof(APOINT)));

    case PRIM_OBSERVER:
        return(ObjSavePrimNode(fh, obj, sizeof(VIEWPOINT)));
```

```
        case PRIM_LINE:
            if(!ObjSavePrimNode(fh, obj, sizeof(struct line)))
                return(FALSE);
            return(ObjSavePrimData(fh, LINEPTR(obj), LINECNT(obj)));

        case PRIM_TRISET:
        {   TRISET *ts = DESCPTR(obj);

            if(!ObjSavePrimNode(fh, obj, sizeof(TRISET)))
                return(FALSE);
            if(!ObjSavePrimData(fh, LINEPTR(obj), LINECNT(obj)))
                return(FALSE);
            if(fwrite((char *)ts->faces, sizeof(FACE), ts->facenum, fh)<ts-> ↩
               facenum)
            {   RealError(ERR_WRITE);
                return(FALSE);
            }
            return(TRUE);
        }
        case PRIM_COORD:
            return(ObjSavePrimNode(fh, obj, sizeof(COORDSYS)));

        case PRIM_GROUP:
            return(ObjSaveGroup(fh, obj));

        default:
            RealError(ERR_INTERNAL);
            return(FALSE);
    }
    return(TRUE);
}

int ObjSaveLevel(FILE *fh, OBJECT *obj)
{
    for(; obj; obj = obj->next)
        if(ObjSave(fh, obj) == FALSE)
            return(FALSE);
    return(TRUE);
}

/*
 * Save given object to a given file
 */

int ObjSave(FILE *fh, OBJECT *obj)
{
    if(obj)
    {   if(ObjSaveNode(fh, obj) == FALSE)
            return(FALSE);
        if(obj->tags)
            if(TagSave(fh, obj->tags) == FALSE)
                return(FALSE);
        if(obj->sub)
          if(ObjSaveLevel(fh, obj->sub) == FALSE)
              return(FALSE);
        return(TRUE);
    }
```

```
      return(FALSE);
  }
```

TAG WRITER EXAMPLE

```
  int TagSave(FILE *fh, RTAG *tl)
  {
      long sl;

      for(; tl; )
      {
    if(tl->tag_ide == RTAG_END) /* 0L */
          {   if(fwrite((char *)tl, sizeof(RTAG), 1, fh) != 1)
                   return(FALSE);
              return(TRUE);
    }
    else
    {
              if(fwrite((char *)tl, sizeof(RTAG), 1, fh) != 1)
                  return(FALSE);

              switch(RTAG_TYPECHAR(tl->tag_ide) /* first byte */
              {
                  case RTAG_INT:
                      if(fwrite((char *)&tl->tag_val,sizeof(long),1,fh)!=1)
                          return(FALSE);
                      break;

                  case RTAG_STR:
                      sl = strlen((char *)tl->tag_val)+1;
                      if(fwrite((char *)&sl, sizeof(long), 1, fh) != 1)
                          return(FALSE);
                      if(fwrite((char *)tl->tag_val, sl, 1, fh) != 1)
                          return(FALSE);
                      break;

                  case RTAG_VEC:
      case RTAG_DVE:
      case RTAG_MVE:
                      if(fwrite((char *)tl->tag_val,sizeof(VECTOR),1,fh)!=1)
                          return(FALSE);
                      break;

                  case RTAG_FLT:
                      if(fwrite((char *)tl->tag_val,sizeof(float),1,fh)!=1)
                          return(FALSE);
                      break;

                  default:
                      RealError(ERR_INTERNAL);
                      return(FALSE);
              }
    }
          tl++;
          break;
```

```
      }
```

1.4 rmtr

```
MATERIAL DATA

The RMTR chunk defines a Real 3D v.2 material. To store a collection of
materials, write several RMTR chunks:

    RMTRxxxx
        struct r3material
    RMTRxxxx
        struct r3material
    ...

If you want to store tags with the material, set the last field
of r3material to a nonzero value and then write the tags to the RMTR
chunk after the material.

The internal structure of RMTR chunk is described below. For future
compatibility, set all reserved fields to zero. The purpose of each
field is not fully explained in this document; see the reference section
of the program manual for further details.

/*
 * Mapping methods - what property is mapped. You may activate several
 * at the same time.
 */

#define MAP_COLOR       (1L<<1)
#define MAP_BUMP        (1L<<2)
#define MAP_BRILL (1L<<3) /* 'Mirrorlike' property mapping */
#define MAP_TRANSP      (1L<<4)
#define MAP_CLIP  (1L<<5)
#define MAP_SHADOW      (1L<<6) /* Intensity filter */

/*
 *  Handler types (see material fields scopehandler, bumphandler etc.)
 */

#define TYPE_DEFAULT    0   /* Real 3D default handler */
#define TYPE_FORMULA  1   /* RPL formula (= eval string) */
#define TYPE_RPL        2   /* RPL procedure */
#define TYPE_CUSTOMBASE 3   /* Custom types start here */

/*
 *  Material flags:
 */

#define MAT_ZEROCOL     (1L<<0)  /* color 0 is transparent */
#define MAT_UNSHADED    (1L<<1)  /* not affected by light srcs (luminous) */
#define MAT_TILEX       (1L<<2)
#define MAT_TILEY       (1L<<3)
#define MAT_FLIPX       (1L<<4)  /* mirrored in horizontal direction */
#define MAT_FLIPY       (1L<<5)
```

```
#define MAT_GRADEX      (1L<<6)  /* color interpolation */
#define MAT_GRADEY      (1L<<7)
#define MAT_SPLINEMAP   (1L<<8)  /* Use natural surface coordinates */
#define MAT_SCOPEMASK   (1L<<9)  /* 0 scope for unpainted points */

#define MAT_EXCLUSIVE   (1L<<13) /* replaces other materials */
#define MAT_SMOOTH    (1L<<14) /* smooth gadget on = no reflections */


/*
 * DOS string length:
 */
#define LEN_DOSNAME        256

/*
 *  Maximum/minimum value for material properties.
 */
#define MAT_MINVAL  0
#define MAT_MAXVAL  100


struct r3material
{   struct R3Node node; /* reserved 14 bytes (set zero) */
    char name[16];
    WORD specularity;   /* From MAT_MINVAL to MAT_MAXVAL */
    WORD specbright;    /* Same as above */
    WORD brilliancy;
    WORD transparency;
    WORD turbidity;
    WORD refraction;    /* rel. speed of light: MAT_MAXVAL = no refraction */
    WORD res1;          /* reserved */
    WORD effectiveness;         /* effectiveness = scope weight in blending */
    WORD res2;                  /* reserved */
    WORD roughness;             /* reflects light to random. direction */
    UWORD flags;                /* see material flags above */
    WORD turbidpower;           /* Turbidity saturation factor */
    UWORD method;                   /* see mapping methods MAP_xxx above */
    UWORD res3;                     /* reserved */
    UWORD txt_freqx;                /* finite horizontal tiling reps */
    UWORD txt_freqy;                /* finite vertical tiling reps */
    float sp_x, sp_y, sp_w, sp_h;   /* spline map offsets & scaling factors*/
    char handler_prg[LEN_DOSNAME];  /* DOS name for proc.handler initializer */
    char picture[LEN_DOSNAME];      /* DOS name for texture map image */
    COLOR transpcolor;              /* used when MAT_ZEROCOL is active */
    WORD bumph;                     /* bump mapping height scale */
    WORD ditherscale;               /* for material color dithering */

    UWORD scopehandler;       /* handler type for effectiveness */
    char scopeexpr[LEN_DOSNAME];    /* RPL handler word or eval. string */
    float scopecon1, scopecon2;     /* constants a & b for handlers */

    UWORD tcoorhandler;             /* customizations in mapping */
    char tcoorexpr[LEN_DOSNAME];
    float tcon1, tcon2;

    UWORD bumphandler;              /* mathematical bumps */
    char bumpexpr[LEN_DOSNAME];
```

```
    float bumpcon1, bumpcon2;

    UWORD colorhandler;             /* mathematical colorings */
    char colorexpr[LEN_DOSNAME];
    float colorcon1, colorcon2;

    UWORD indexhandler;             /* texture map indexing */
    char indexexpr[LEN_DOSNAME];
    float indexcon1, indexcon2;

    LONG reserved[8];
    LONG tags;                      /* Tags follow, if nonzero */
};
```

1.5 rvrs

```
    This is always the first chunk in the file, and should be always
    included.

  struct r3version
  {   long platform;      /* ('A'<<24) | ('M'<<16) | ('I'<<8) | 'G' */
            long version;       /* set to 2 */
            long revision;      /* set to 20 */
        };
```

1.6 display driver

```
                INTERFACE FROM REAL 3D TO CUSTOM DISPLAY DEVICES


This file contains support material and information necessary when
creating a shared library for interfacing frame buffers, graphics
adapters etc. custom display devices from Real 3D v.1.4 and v.2.

The file contains information of how to create the library and how to
use Real 3D's features which are related to custom display devices.

An example implementation of the interface library ( for Harlequin ),
written in SAS C, is included. This example can be used freely.


CONTENTS
--------


                 General notes

                 The user interface for the custom display

                 The library interface for the custom display

                 Creating the interface library
```

1.7 general

GENERAL NOTES

It is possible to connect a custom display device (frame buffer etc.)
to Real 3D so that images can be rendered directly to the custom display.
This kind of direct support can be obtained by creating a simple shared
library containing some elementary functions.

1.8 userinterface

THE USER INTERFACE FOR THE CUSTOM DISPLAY

The custom display is controlled using a set of menu functions in the
Real 3D's user interface. These menu functions can be found from the menu
'Project/External screen' and they are:

Settings
This function allows the user to define the name of the desired shared
library and the file format which is used when saving images from
the custom display.

Open screen
Tries to obtain access to the custom display. If opening is successful,
4 bit rendering can be directed to the custom display. The function also
sets default resolution and display modes.

Close screen
Frees the custom display. For example, if the access is exclusive (no
multiple screens supported), other multitasking programs can access the
display after this.

Set modes
Allows the user to select desired custom display modes, such as interlace,
hires, resolution etc. All the options available in the function depend on
the device used.

Save
This function saves the custom display image by reading the image line by
line and saving it using the specified file format, either IFF24 or Targa
or a custom format. Custom format saving can be used to save the custom
display image in its native format instead of the standard 24 bit formats.
For example, if the custom display is not a true 24 bit display, there are
probably more memory efficient ways to save the image. Also, the function

can be used for faster image saving, because line based data transfer in the standard 24 bit save function inevitably requires extra data moving and format transformations, and is consequently somewhat slow. Nevertheless, a proper implementation of custom format saving is optional; the library may contain a non-operational function instead.

In addition to the previous functions, most of the other rendering controls (Render settings/Aspect ratio, image size etc.) affect the custom display.

An example of custom display usage from Real 3D v.2:

- Install the custom display hardware and its driver software to your system.

- Copy the R3D-custom display interface library to LIBS: directory. The name of this library is usually of the form xxx_r3d.library.

- Select the menu 'Project/External screen/Settings' in Real 3D Then define the name of of the interface library and select OK.

- Select 'External screen/Open'.

- You may now try to change the custom display resolution and other such properties using the External screen/Set modes function.

- Activate a view and select 'View/Render/Settings. Then set the 'Output' cycle gadget to 'External'. If the custom display was successfully opened, the image resolution fields 'Width' and 'Height' are automatically changed to show the correct dimensions. If not, repeat the previous steps to ensure proper installation and usage.

- Select OK and then View/Render/Window. An image should start to appear to the custom display.

- When the image is ready, you can save it by selecting 'External screen/ save'. This asks the name for the image and saves the image in the format specified by using 'External screen/Settings' function. You may try to save the image using the custom format, but it is possible that the function is not operational. Nevertheless, it is harmless to try it.

- When you do not need the custom display any longer, close it using the External screen/Close menu (of course, the screen is automatically closed when the user exits the program).

- For further details of the Custom save format and the features included in the Set modes functions, read the readme file of the interface library of your custom display device. This file should contain all the hardware specific details.

1.9 libraryinterface

THE LIBRARY INTERFACE FOR THE CUSTOM DISPLAY

The shared library needed to connect a display hardware product to Real 3D must contain the following functions:

```
R3DInitDspDrv()
R3DFreeDspDrv(handle)
R3DSetMode(handle)
R3DGetSize(handle,x,y)
R3DWriteLine(handle,buffer,len,x,y)
R3DReadLine(handle,buffer,len,x,y)
R3DClsScr(handle,col)
R3DCustomSave(handle,name,x,y,w,h)
```

Furthermore, another set of additional functions may be included:

```
R3DInitRen(handle)
R3DEndRen(handle)
R3DEndRow(handle,y)
R3DGetAspect(handle,w,h)
R3DExtF(handle)
```

These extra functions are used by Real 3D v.2 providing that the library version identifier equals 40. This library extension possibility was added in order to make it easier to write the library for graphics boards which do not use separate monitor.

Function descriptions:

All the integers in the following descriptions are 32 bit wide.
The library code is run by a process.

*** NAME

R3DInitDspDrv -- allocate and initialize a screen on the custom display

SYNOPSIS

```
int R3DInitDspDrv()
```

FUNCTION

Allocates and initialises a custom display and sets default display formats.

The main purpose of this function is to get access to the custom display. After the call, the custom display should be ready to receive and send data, to be cleared and to other such manipulations needed when rendering images to the custom display.

For example, this function may open the actual library for the display device and then open a screen.

After the function, Real 3D asks the resolution for the custom display using the R3DGetSize() and resets image size to this

```
  resolution.

    INPUTS

    RESULT
        A non-zero value, if the call was successful. The value is used
        as a handle, with which the custom display is accessed. Real 3D
        does not utilize multiple custom display screens yet, but this
        may change in the future versions.

    SEE ALSO
        R3DFreeDspDrv(), R3DSetMode(), R3DGetSize()


*** NAME
        R3DFreeDspDrv -- closes a custom display previously opened
        by a call to R3DInitDspDrv

    SYNOPSIS
        void R3DInitDspDrv(int handle)
                             A0

    FUNCTION
        Frees the custom display access. For example, it may close a
        screen and then close the actual library for the display device.

    INPUTS
        handle -- a handle obtained by a call to R3DInitDspDrv

    RESULT

    SEE ALSO
        R3DInitDspDrv()


*** NAME
        R3DSetMode -- Asks and sets display modes, resolution etc.
        properties for the custom display

    SYNOPSIS
        void R3DSetMode(int handle)
                          A0

    FUNCTION
        This function is used to select hardware specific options for
        the custom display device. It should request all the relevant
        options available for the device such as INTERLACE, PAL/NTSC,
        full screen resolution alternatives, display depth etc.

        The mode request may be done for example by opening a requester
        on the IntuitionBase->ActiveWindow and by monitoring the UserPort
        of this window. Real 3D's rendering control screen window is
        guaranteed to contain GADGETUP class IDCMP messages. Nevertheless,
        any other screen or method may be used as well. Also, the function
        may be non-operational, if only one display mode and size is
        available.
```

```
        After getting the mode selections from the user, the custom display
        defined by handle should be modified to fulfil the selections;
  the current image may be cleared, trashed or preserved. After the
  function, Real 3D asks the new full screen resolution
        using the R3DGetSize() function.

    INPUTS
        handle -- a handle obtained by a call to R3DInitDspDrv

    RESULT

    SEE ALSO
        R3DGetSize()


*** NAME
        R3DGetSize -- Asks the current full screen resolution of the
        custom display

    SYNOPSIS
        void R3DGetSize(int handle,int *x,int *y)
                         A0         A1     D0

    FUNCTION
        This function is used to find out the full screen resolution
        of the custom display after opening it and after mode selections.

    INPUTS
        handle -- a handle obtained by a call to R3DInitDspDrv
        x      -- a pointer to 32 bit integer to which the horizontal
                  resolution is written
        y      -- a pointer to 32 bit integer to which the vertical
                  resolution is written

    RESULT

    SEE ALSO
        R3DSetMode()


*** NAME
        R3DWriteLine -- Writes a line of rgba data into the custom display

    SYNOPSIS
        void R3DWriteLine(int handle,int *buffer,int len,int x,int y)
                           A0         A1          A2      D0    D1

    FUNCTION
        This function writes one line of data, pointed by the parameter
        buffer, into the custom display. The data is in the form
        rgba rgba ..., four consecutive bytes for each pixel. The fourth
        byte 'a' is for alpha channel; this is not utilized by the current
        Real 3D version, but this may change in the future versions. The
        values for each color channel have range 0-255.

        The position of the target line on the custom display is defined
        by the offset parameters x and y, and the length of the line is
```

```
        defined by the parameter len.

    INPUTS
        handle -- a handle obtained by a call to R3DInitDspDrv
        buffer -- points to 4*len bytes of rgba data to be written
        len    -- the number of pixels to be written
        x, y   -- offset values from the top left corner

    RESULT

    SEE ALSO
        R3DReadLine()


*** NAME
        R3DReadLine -- Reads a line of rgba data from the custom display
        to a buffer

    SYNOPSIS
        void R3DReadLine(int handle,int *buffer,int len,int x,int y)
                          A0         A1          A2      D0    D1

    FUNCTION
        This function reads the color of len pixels into buffer. The data
        is written in the form rgba rgba ..., four consecutive bytes for
        each pixel. The fourth byte 'a' is for alpha channel; this is not
        utilized by this Real 3D demo version, (Real 3D v.2 supports
        Alpha channel). The values for each color channel should be scaled
        to full range 0-255.

        The data is read starting from the position (x,y), and the length
        of the line is defined by the parameter len.

    INPUTS
        handle -- a handle obtained by a call to R3DInitDspDrv
        buffer -- points to 4*len bytes of buffer area
        len    -- the number of pixels to be read
        x, y   -- offset values from the top left corner

    RESULT

    SEE ALSO
        R3DWriteLine()


*** NAME
        R3DClsScr -- Clears the custom display

    SYNOPSIS
        void R3DClsScr(int handle, int color)
                        A0          A1

    FUNCTION
        This function clears the color of the custom display to a specified
        color.  The color is defined by the parameter 'color' in the form
        rgba, one byte for each color channel.
```

```
    INPUTS
        handle -- a handle obtained by a call to R3DInitDspDrv
        color  -- the background color of the display after the call

    RESULT

    SEE ALSO
        R3DWriteLine()


*** NAME
        R3DCustomSave -- Saves the custom display image

    SYNOPSIS
        int *R3DCustomSave(int handle,char *name,int x,int y,int w,int h)
                             A0         A1         A2    D0    D1    D2

    FUNCTION
        This function may be used to save a part of the custom display
        contents. The functionality of R3DCustomSave() is optional,
        because standard 24 bit saving facilities of Real 3D are
        available.

        There are two reasons to implement the function: first of all,
        the speed, and secondly, custom formats. There are several display
        devices available, whose picture information can be stored more
        efficiently than when using 24 bit formats.

        With the v.40 library implementations, the function return value
        is considered by Real 3D. If the value is non-zero, Real 3D
        interprets the value as a pointer to a ViewPort and saves the
        ViewPort contents using its standard IFF saving routines. This
        feature can be utilized by graphics extensions whose data
        structures are identical with standard Amiga display structures.
        Anyway, it should be noted that it is not guaranteed that the
        saving routines can save possible future expansions of Amiga
        Viewport modes.
        NULL value must be returned, if ViewPort saving is not wanted.

    INPUTS
        handle -- a handle obtained by a call to R3DInitDspDrv
        name   -- null-terminated character string for the image file name
        x,y    -- the top-left corner of the sub-image to be saved
        w,h    -- the size of the sub-image to be saved

    RESULT
        NULL or a pointer to a ViewPort.

    SEE ALSO


*** NAME
        R3DInitRen -- Initialize display for rendering of a new image

    SYNOPSIS
        int R3DInitRen(int handle)
                         A0
```

```
    FUNCTION
        This function is always called when starting to render an image,
        and therefore it can be used for device dependent preparations.
        After the call, the custom display should be ready to receive
        consecutive R3DWriteLine() data lines.

        The main purpose of the function is to 'inform' the custom display
        system about starting the rendering. For example, in one monitor
        systems, this function can bring the custom display to the front.

    INPUTS
        handle -- a handle obtained by a call to R3DInitDspDrv

    RESULT
        TRUE (=1) if initialization is successful. Returning 0
        cancels rendering.

    SEE ALSO
        R3DEndRen()


*** NAME
        R3DEndRen -- Actions required after rendering

    SYNOPSIS
        void R3DEndRen(int handle)
                           A0

    FUNCTION
        This function is always called when rendering of an image is
        finished/cancelled.

        The main purpose of the function is to 'inform' the custom
        display system about ending the rendering. For example, in one
        monitor systems, this function may put the custom display to
        the back.

    INPUTS
        handle -- a handle obtained by a call to R3DInitDspDrv

    RESULT

    SEE ALSO
        R3DInitRen()


*** NAME
        R3DEndRow -- Actions required after rendering a row

    SYNOPSIS
        int R3DEndRow(int handle, int y)
                          A0       A1

    FUNCTION
        This function is called after finishing each row.
```

```
        The main purpose of the function is to inform the custom
        display system about ending the current row.

        The return value 0 cancels the rendering. This can be used
  to stop the rendering, for example because of detecting an error
  condition or an user action requiring cancelling.

    INPUTS
        handle -- a handle obtained by a call to R3DInitDspDrv
  y = the vertical position of the current row

    RESULT
  FALSE (=0) cancels rendering
  TRUE (=1) continues rendering

    SEE ALSO


*** NAME
        R3DGetAspect -- Asks the current pixel aspect ratio of the
        custom display

    SYNOPSIS
        void R3DGetAspect(int handle, int *w, int *h)
                             A0          A1      D0

    FUNCTION
        This function is used to find out the pixel aspect ratio.
  The ratio is described using two integers, which define the
  proportion of the pixel edges. For example, hires-noninterlaced
  ratio can be usually defined by *w = 1, *h = 2.

    INPUTS
        handle -- a handle obtained by a call to R3DInitDspDrv
        w      -- a pointer to 32 bit integer to which the pixel
                  width is written
        h      -- a pointer to 32 bit integer to which the pixel
                  height is written

    RESULT

    SEE ALSO


*** NAME
        R3DExtF -- Extension function

    SYNOPSIS
        int R3DExtF(int handle)
                        A0

    FUNCTION
  An extension function for future library modifications.

    INPUTS
        handle -- a handle obtained by a call to R3DInitDspDrv
```

```
    RESULT
  Must return 0

    SEE ALSO
```

1.10 creatinginterface

```
CREATING THE INTERFACE LIBRARY FOR REAL 3D


The functions of the previous chapter may be implemented quite freely.
For example, there are no special requirements for mode query, custom
saving, multiple screens or alpha channel usage.

The library may be created quite easily by modifying the example code
contained in this disk; the rather primitive but sufficient example is
for interfacing ACS Harlequin frame buffer.

Note that in this latest specification we require exact library version
numbering. If all the functions described above, including the v.40
library extension functions R3DInitRen - R3DExtF are needed in your
library implementation, the library version number must be 40
(Revision number is unused by Real 3D). If extension functions are
not included, the version number should be less than 40.

We have used 'xxx_r3d.library' naming convention for interface
libraries, but this is optional.

The interface library should be documented with a text file, which
describes the user-visible features of the library, that is, the options
available in the Set modes function and the Custom save format usage.

If the function collection documented here seems not to be sufficient,
please inform Realsoft of the new functions required, so that they can be
added to the end of the library, and relevant calls can be added into
the program itself. Also, if this document contains insufficient amount
of information for creating the library, please do not hesitate contact us:

    Realsoft KY
    Attn. Vesa Meskanen
    KP 9, SF-35700 Vilppula
    Finland

    tel. +358-34-4718390
    fax  +358-34-4718533
```

1.11 changesv40

```
V.40 CHANGES


If you have already written an interface library using the original
```

```
v.34 library specification, please note the following changes:

- The function R3DClsScr() is no longer called whenever the rendering
starts, because Real 3D v.2 supports subsquare rendering to external
display. Instead, a function R3DInitRen() is called as a sign of
starting the rendering.

- The return value of R3DCustomSave() is now considered (previously
void ). If it is not NULL, this is taken as a request to save an Amiga
Viewport using Real 3D's IFF saving routines. Therefore, a proper return
value should be always returned.

- Five new functions have been added to the end of the library, for
easier display control.

- The new libraries using extended function set must have version
number 40.
```

1.12 examplecode

```
EXAMPLE CODE


**** This is the link file


libfd /include/dspdrv.fd
TO       libs:hr_r3d.library
FROM     LIB:libent.o lib:libinit.o hrcode.o
libVersion 40 libRevision 1


***** The .fd file:


##base _R3DDspDrvBase
##bias 30
##public
R3DInitDspDrv()
R3DFreeDspDrv(hnd)(A0)
R3DSetMode(hnd)(A0)
R3DGetSize(hnd,x,y)(A0/A1/D0)
R3DWriteLine(hnd,col,c,x,y)(A0/A1/A2/D0/D1)
R3DReadLine(hnd,col,c,x,y)(A0/A1/A2/D0/D1)
R3DClsScr(hnd,col)(A0/A1)
R3DCustomSave(hnd,name,x,y,w,h)(A0/A1/A2/D0/D1/D2)
R3DInitRen(hnd)(A0)
R3DEndRen(hnd)(A0)
R3DEndRow(hnd,y)(A0/A1)
R3DGetAspect(hnd,x,y)(A0/A1/D0)
R3DExtF(hnd)(A0)
##end


/*************************************************************\
```

```
 This file contains the library version 40 functions for ACS Harlequin.
 In this example alpha channel properties of Harlequin are used, but
 no multiple screens are supported. Therefore, the handle argument
 and the internal 'hscreen' pointer have the same value, and hence
 most functions ignore the 'hnd' argument and use 'hscreen' instead.
 I hope this is not confusing.

 Custom saving is not implemented.

 It is quite easy to modify the functions to fit the properties of
 another similar device. To recompile the modified file, use the
 command

    sc SAVEDS NOSTKCHK LIBCODE hrcode' (SAS C v.6.2)

\*****************************************************************/


#include <exec/types.h>
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#include <libraries/gadtools.h>
#include <proto/all.h>
#include <libraries/harlequin.h>
#include <proto/harlequin.h>

struct ExecBase      *SysBase;
struct DosLibrary    *DOSBase;
struct ConfigDev     *ConfigDev;
struct Library       *ExpansionBase;
struct IntuitionBase *IntuitionBase;
struct Library       *GadToolsBase;
struct GfxBase       *GfxBase;
struct Library       *HarlequinBase;


/* Harlequin related data items: */


struct HNewScreen myscreen =
    {0,0,740,576,HINTERLACE|HPAL|HBORDERLESS|HNEAREST,-1,0,0};
struct HImageBlock hbl = {0,1,IB_RED|IB_GREEN|IB_BLUE|IB_ALPHA,};

struct HScreen *hscreen;



void __saveds __UserLibCleanup()
{   if(GfxBase)       CloseLibrary((struct Library *)GfxBase);
    if(ExpansionBase) CloseLibrary((struct Library *)ExpansionBase);
    if(IntuitionBase) CloseLibrary((struct Library *)IntuitionBase);
    if(DOSBase)       CloseLibrary((struct Library *)DOSBase);
    if(SysBase)       CloseLibrary((struct Library *)SysBase);
    if(GadToolsBase)  CloseLibrary((struct Library *)GadToolsBase);
    if(HarlequinBase)  CloseLibrary((struct Library *)HarlequinBase);
}
```

```
int __saveds __UserLibInit()
{
    if((SysBase = (struct ExecBase *)
        OpenLibrary ("exec.library",0)) == NULL) goto error;
    if((DOSBase = (struct DosLibrary *)
        OpenLibrary ("dos.library",0)) == NULL) goto error;
    if((ExpansionBase = (struct ExpansionBase *)
        OpenLibrary ("expansion.library",0)) == NULL) goto error;
    if(!(IntuitionBase=(struct IntuitionBase *)
      OpenLibrary("intuition.library",0))) goto error;
    if(!(GadToolsBase=(struct Library *)OpenLibrary("gadtools.library",0)))
        goto error;
    if(!(GfxBase = (struct GfxBase *)OpenLibrary("graphics.library",0)) )
  goto error;
    if(!(HarlequinBase = OpenLibrary("harlequin.library",0L)))
        goto error;
    return(0);
  error:
    __UserLibCleanup();
    return(1);
}

/* Library functions: */

R3DInitDspDrv()
{   if(hscreen) return(NULL); /* Exclusive access through this library !!*/

    if(hscreen = HOpenScreen(&myscreen))
        HScreenFunction(hscreen,SCREEN_ON|SCREEN_FRONT);
    else return(NULL);

    return(hscreen);
}

void __asm R3DFreeDspDrv(register __a0 void *hnd)
{   if(hscreen) HCloseScreen(hscreen);
    hscreen = NULL;
}

void __asm R3DWriteLine( /* writes rgb of pixnum pixels */
register __a0 int hnd,
register __a1 UBYTE *buf,
register __a2 int pixnum,
register __d0 int x,
register __d1 int y)
{   if(!hscreen) return;
    hbl.Data = buf; hbl.Width = pixnum;
    HClipBlockScreen(&hbl,0,0,hscreen,x,y,pixnum,1);
}

void __asm R3DReadLine(
register __a0 int hnd,
register __a1 UBYTE *buf,
register __a2 int pixnum,
register __d0 int x,
register __d1 int y)
{   if(!hscreen) return;
```

```
    hbl.Data = buf; hbl.Width = pixnum;
    HClipScreenBlock(hscreen,x,y,&hbl,0,0,pixnum,1);
}

void __asm R3DClsScr(register __a0 int hnd, register __a1 ULONG color)
{   if(!hscreen) return;
    hscreen->FgPen = color; HClearScreen(hscreen);
}

void __asm R3DGetSize(
register __a0 int hnd,
register __a1 int *x,
register __d0 int *y)
{   *x = hscreen->Width; *y = hscreen->Height;
}



struct TextAttr myfont =
{   "topaz.font", /*  Name */
    8,      /*  YSize */
    0,      /*  Style */
    0,      /*  Flags */
};




#define WND_WIDTH   260
#define WND_HEIGHT   90

#define GAD_CANCEL  1
#define GAD_OK      2

void __asm R3DSetMode(register __a0 int hnd)
{   struct IntuiMessage *message;
    struct Window *window=NULL;
    ULONG class;

    struct TextFont *RealFont;
    void *vi = NULL;
    UWORD topborder;

    struct Gadget *gad, *glist=NULL, *hiresgadget, *lacegadget;
    struct NewGadget ng;
    struct Screen *scr;

    if (!(RealFont = (struct TextFont *)OpenFont(&myfont)))
        return;

    scr = LockPubScreen(0);
    if(!scr || (WND_WIDTH > scr->Width || WND_HEIGHT > scr->Height))
    {   if(scr) UnlockPubScreen(NULL, scr);
        scr = LockPubScreen("Workbench");
    }
    if(scr) UnlockPubScreen(NULL, scr);
```

```
  else goto bail_out;
  ScreenToFront(scr);

  if(!(vi = GetVisualInfo(scr, TAG_DONE))) goto bail_out;
  topborder = scr->WBorTop + (scr->Font->ta_YSize + 1);
  if(!(gad = CreateContext(&glist))) goto bail_out;

  ng.ng_LeftEdge = WND_WIDTH/2;
  ng.ng_TopEdge = 4+topborder;
  ng.ng_Width = 0;
  ng.ng_Height = 14;
  ng.ng_GadgetText = "Select display modes";
  ng.ng_TextAttr = &myfont;
  ng.ng_GadgetID = 0;
  ng.ng_Flags = PLACETEXT_IN | NG_HIGHLABEL;
  ng.ng_VisualInfo = vi;
  gad = CreateGadget(TEXT_KIND, gad, &ng,
TAG_DONE);

  ng.ng_TopEdge += 20;
  ng.ng_LeftEdge = WND_WIDTH-40;
  ng.ng_Width =28;
  ng.ng_Height = 14;
  ng.ng_Flags = NULL;
  ng.ng_GadgetText = "Interlace";
  gad = lacegadget = CreateGadget(CHECKBOX_KIND, gad, &ng,
GTCB_Checked, hscreen->Type & HINTERLACE,
TAG_DONE);

  ng.ng_TopEdge += 20;
  ng.ng_GadgetText = "Hires";
  gad = hiresgadget = CreateGadget(CHECKBOX_KIND, gad, &ng,
GTCB_Checked, hscreen->Width  == 910,
TAG_DONE);

  ng.ng_TopEdge += 20 ;
  ng.ng_LeftEdge = 14;
  ng.ng_Width = 80;
  ng.ng_GadgetText = "OK";
  ng.ng_GadgetID = GAD_OK;
  gad = CreateGadget(BUTTON_KIND, gad, &ng, TAG_DONE);

  ng.ng_LeftEdge = WND_WIDTH - 80 - 14;
  ng.ng_GadgetText = "CANCEL";
  ng.ng_GadgetID = GAD_CANCEL;
  gad = CreateGadget(BUTTON_KIND, gad, &ng,
TAG_DONE);

  if (!(window = OpenWindowTags(NULL,
    WA_Width, WND_WIDTH, WA_InnerHeight, WND_HEIGHT,
    WA_IDCMP, GADGETUP, WA_CustomScreen, scr,
      WA_DragBar, TRUE,
WA_DepthGadget, TRUE,
      TAG_DONE)))
    goto bail_out;

  AddGList(window, glist, ((UWORD) -1), ((UWORD) -1), NULL);
```

```
    RefreshGList(glist, window, NULL, ((UWORD) -1));
    GT_RefreshWindow(window, NULL);

    for(;;)
    {   while(!(message=(struct IntuiMessage *)GetMsg(window->UserPort)))
            Wait(1<<window->UserPort->mp_SigBit);
        if(message)
        {   class = message->Class;
            gad = (struct Gadget *)message->IAddress;
            ReplyMsg((struct Message *)message);
            if(class == GADGETUP && gad->GadgetID) break;
        }
    }
    if(gad->GadgetID == GAD_OK)
    {   if(lacegadget->Flags & SELECTED) myscreen.Type |= HINTERLACE;
  else myscreen.Type &=~HINTERLACE;
  if(hiresgadget->Flags & SELECTED) myscreen.Width = 910;
  else myscreen.Width = 740;
        HConvertScreen(hscreen,&myscreen,0);
    }
   bail_out:
    if(window) CloseWindow(window);
    if(vi) FreeVisualInfo(vi);
    if(glist) FreeGadgets(glist);
    if(RealFont) CloseFont(RealFont);
}

int __asm R3DCustomSave(
register __a0 int hnd,
register __a1 char *name,
register __a2 int x,
register __d0 int y,
register __d1 int w,
register __d2 int h)
{   return(NULL); /* Must return NULL, unless Amiga Viewport save is wanted */
}


/* Extended library v.40 functions. They are actually not necessary
with Harlequin, but they are included here as an example. */

int __asm R3DInitRen(register __a0 int hnd)
{   return(TRUE);
}

void __asm R3DEndRen(register __a0 int hnd)
{
}

int __asm R3DEndRow(register __a0 int hnd, register __a1 int y)
{   return(TRUE);
}

void __asm R3DGetAspect(
register __a0 int hnd,
register __a1 int *w ,
register __d0 int *h)
```

```
{    *h = *w = 1;
}

int __asm R3DExtF(register __a0 int hnd)
{    return(NULL);
}
```