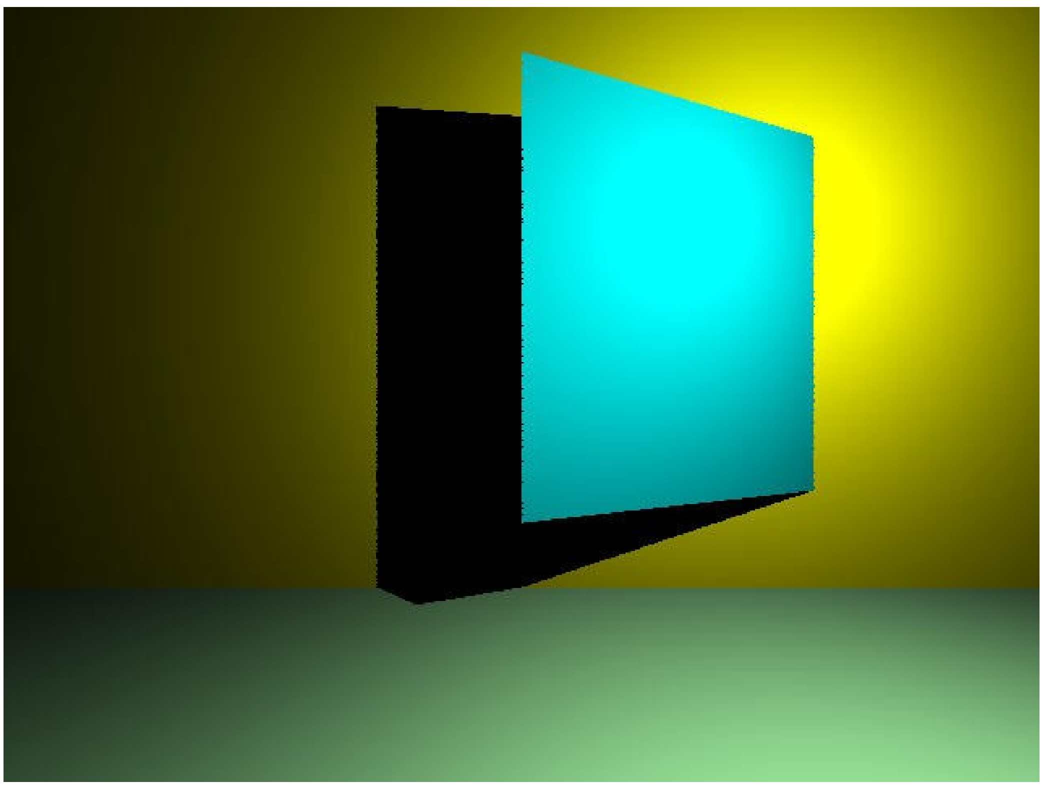
Finite rectangular planes.

The planes that we have previously used have all been of unbounded size. We can also define a finite or bounded plane object.

The *point* field used in the definition of an unbounded plane represents the location of the "lower left" corner of the finite, rectangular plane. The plane *normal* plays its usual role. Additional quantities are required:

> The vector *xdir[3]* *when projected into the plane* and represents the direction of increase of the *x* coordinate. The vector should be projected and converted into a unit vector either at the time the finite plane definition is loaded or at the time the finite plane description is dumped to the *stderr* log.
>
> The vector *xsize[2]* vector provides the *width* and *height* of the rectangular plane in the *x* and *y* directions respectively.



Data structures for the finite plane

In the C language, there is no "built in" support for derived classes and inheritance, but, as seen earlier, we can build it in ourselves, by adding a *priv* pointer the the *plane type.*

```
typedef struct plane_type {
   double  normal[3];
     :
   void    *priv;         /* Data for specialized types  */
}  plane_t;
```

```
typedef struct fplane_type {
   double  xdir[3];        /* x axis direction   */
   double  size[2];        /* width x height     */
   double  rotmat[3][3];   /* Rotation matrix    */
   double  lasthit[2];     /* used for textures */
} fplane_t;
```

Alternatively (but more easily) one could just junk up the *plane_t* structure with finite plane and/or textured plane attributes.

Input Data for the finite plane

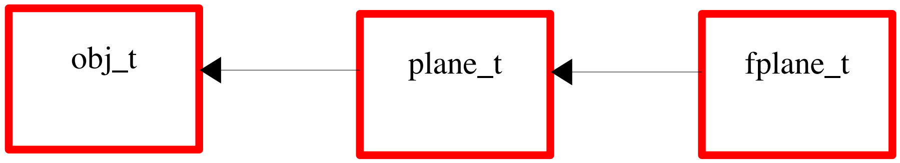
```
15             finite plane
0  0  0        r g b ambient
0  6  6        r g b diffuse
0  0  0        r g b specular

1   0   3      normal
4   1  -3      point

1   2   0      x direction
5   5          size
```

Initializing the *fplane_t*

In a true object oriented language we would have the following inheritance structure



When a new plane_t was created, constructors for *obt_t, plane_t,* and *fplane_t* would be *automatically* activated in that order. As we have seen there are no constructors for C structures, but we can emulate the behavior through explicit calls and *avoid needless duplication of code.*

```
obj_t *obj_init(...) {
     allocate new obj_t;
     link it into the object list;
     return(obj);
}

obj_t *plane_init(...) {
    obj  = obj_init(...);
    allocate new plane_t;
    link it to obj->priv;
    load material properties
    load plane geometry
    return(obj);
}

obj_t *fplane_init(...) {
   obj = plane_init(...);
   pln = obj->priv;
   allocate new fplane_t;
   link it to pln->priv;
   read xdir, size;
   project xdir onto infinite plane
   compute required rotation matrix
   return(obj);
}
```

An analogous strategy can be used in *fplane_dump().*

The *fplane_hits()* function

The *obj->hits()* pointer for a finite plane should point to *fplane_hits()*.

/ Comment Me */*
```
double  fplane_hits(double  *base, double  *dir, obj_t   *obj);
```

Even though an *fplane_hits()* function is required, it would be a *very bad* idea to paste the internals of *plane_hits()* inline here. Instead, *plane_hits()* should be invoked to determine if and where the ray hits the infinite plane in which the finite plane is contained.

```
     t = plane_hits(base, dir, obj);
     if (t < 0)
          return(t);
```

Arrival here means that the ray hit the infinite plane and that the location of the hit has been stored in *obj->hitloc[],* The next task is to determine *if the hit location lies within the bounds of the prescribed rectangular area.*

In general, this seems like a *very difficult* problem. But there is a case for which the answer is simple. Suppose the base of the rectangle happened to be at *(0, 0, 0)*, the *xdir[]* vector was *(1,0.0)* and the plane normal is *(0, 0, 1).* In that case, the rectangular finite plane is based at the origin and lies in the (x, y) plane. Therefore the following test could be applied.

```
   if ((obj->hit[0] > fp->size[0]) || (obj->hit[0] < 0.0))
        return(-1);

   if ((obj->hit[1] > fp->size[1]) || (obj->hit[1] < 0.0))
      return(-1);
```

Transforming the coordinates of the finite plane.

A two-step coordinate system transformation may be applied to the original *obj->hitloc[]* to permit use of the simple test on the previous page:

(1) translate (move) the lower left corner of the finite plane to the origin.
and
(2) rotate the coordinate system so that
the plane normal rotates into the positive Z-axis and
the xdir[] vector rotates into the X-axis

Step 1 can be accomplished via a simple:

```
vec_diff3(pln->point, obj->hitloc, newhit);
```

Constructing the rotation is slightly more complicated. Once the rotation has been constructed the second step may be accomplished via:

```
xform3(rot, newhit, newhit);
```

After this is done, the simple test on the previous page may be applied to *newhit.*

Rotating an arbitrary vector pair of orthogonal vectors into the *x* and z axes

```
int main() {
   double rot[3][3];
   double irot[3][3];
   double norm[3];
   double xdir[3];

   double v1[3];
   double v2[3];
   double v3[3];

/* plane normal */

   norm[0] = 1.0; norm[1] = 1.0; norm[2] = 1.0;

/* the x direction */

   xdir[0] = 1.0; xdir[1] = 0.0; xdir[2] = -1.0;
/*
 * The first row of the rotation matrix will rotate into
 * the x-axis so that's where we put xdir
 */
   unit_vec3(xdir, rot[0]);
/*
 * We want the normal to end up on the z-axis so we make it
 * the third row of the rotation matrix
 */

   unit_vec3(norm, rot[2]);
/*
 * Once two rows of a rotation are set, the third one
 * is automatic... it has to be orthogonal to the
 * other two.
 */
   cross3(rot[2], rot[0], rot[1]);
   mat_prn3("Rotation matrix", rot);
```

```
   /* Demonstrate that the normal rotates into the z-axis */

   xform3(rot, norm, v2);
   vec_prn3("\nRotated normal   ", v2);

   /* and that xdir rotates into the x-axis */

   xform3(rot, xdir, v3);
   vec_prn3("Rotated xdir     ", v3);

  /* The inverse of a rotation is its transpose */

   xpose3(rot, irot);
   xform3(irot, v3, v1);
   vec_prn3("Rotated back xdir", v1);
}
```

```
 Rotation matrix

       0.707       0.000      -0.707
      -0.408       0.816      -0.408
       0.577       0.577       0.577

Rotated normal        0.000    0.000    1.732
Rotated xdir          1.414    0.000    0.000
Rotated back xdir      1.000    0.000   -1.000
```