```
/**************************************************************************/
/*                                                                        */
/*  RAY TRACER                                                            */
/*  Copyright (c) 1990 by Lawrence Kesteloot and Fredrik Fatemi           */
/*                                                                        */
/*  Compile on an IRIS computer with:                                     */
/*     cc ray.c -o ray -Zg -Zf                                            */
/*                                                                        */
/*  To generate an image in the background, type:                         */
/*     ray name.world &                                                   */
/*                                                                        */
/**************************************************************************/

#include "comptype.h"  /* This file will tell us what computer this is  */
                       /* being compiled on.  If DOGRAPH is defined,    */
                       /* then all of the graphics commands will be     */
                       /* compiled too.  If it is not defined, then     */
                       /* the graphics commands will be commented out   */
                       /* and the program can run on a VAX or something */

#include "stdio.h"
#include "math.h"
#ifdef DOGRAPH
#include "gl.h"
#include "device.h"
#include "savescreen.h"
#endif

#ifdef DOGRAPH
#define IRIS4D    /* If this is defined, this is begin compiled on a 4D Iris */
#endif

#define PI 3.1415926
#define INITSTEP 16   /* The starting resolution */
#define ENDSTEP 1     /* The final resolution    */
#define GETMAXX 1023  /* Last pixel horizontally */
#define GETMAXY 767   /* Last pixel vertically   */
#define DISTANCE 1.0  /* Distance from eye to screen */
#define WIDTH 0.6     /* Width of the screen.  The distance from the  */
                      /* eye to the screen is 1.0, and this WIDTH     */
                      /* will tell you how much the picture is zoomed */
                      /* in.                                          */

#define FILENAME "SPHERES.SCR"  /* File to which to save the image */

#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
#define TRUE (!FALSE)
#endif

#define pmin (-2.0)  /* The next five are the parameters for the Mandelbrot */
#define pmax 1.0     /* set which is one of the surfaces available.         */
#define qmin (-1.5)
#define qmax 1.5
#define MAXCOUNT 100  /* Max number of iterations for the Mandelbrot set */

#define CHECKERBOARD 0 /* Constants for the surface */
#define MANDELBROT 1
#define WATER 2        /* WATER does not work yet */

#define RIPPLE 0.04  /* Frequency of ripples of water */

#define AMP 4        /* Amplitude of the waves */

                /* Constants for bobject.reflect */
#define DULL 0   /* The object does not reflect anything */
#define SHINE 1  /* The object reflects all light rays to it */
#define GLASS 2  /* We can see through the object and the light is refracted */
#define MATTE 3  /* Like DULL but has some diffused shine from the light */

#define SPHEREOBJECT 0  /* Constants for bobject.objecttype */
#define TORUSOBJECT 1
#define CONEOBJECT 2
#define ELLIPSELOIDOBJECT 3
#define PARABOLOIDOBEJECT 4
#define EGGOBJECT 5
#define HYPERBOLOID 6
#define CYLINDER 7

#define SILVER 1    /* Make bobject[i].Red=SILVER to color the object silver */

#ifndef DOGRAPH
typedef unsigned char RGBvalue;
#endif

char *OBJECTNAME[]={"Sphere","Torus","Cone ","Ellipseloid","Paraboloid","Egg  ","Hyperbol
oid","Cylinder"};
char *SURFACENAME[]={"Dull","Shine","Glass","Matte"};
char *PLATFORMNAME[]={"Checkerboard","Mandelbrot","Water"};

struct BOBJECT
  {
    int objecttype;  /* See constants above */
    float x,y,z,r,A,B,C,sina,cosa,tana,top,bottom,ax,ay,az;
    RGBvalue Red,Green,Blue;  /* Color at brightest point */
    int reflect;       /* See constants above */
  };

struct RAY
  {
    float X1,Y1,X2,Y2,Z1,Z2; /* x=X1*t+X2; y=Y1*t+Y2; z=Z1*t+Z2 */
  };

struct BOBJECT bobject[200]; /* 0-99 are objects, and 100-199 are lights */
int inside[200];   /* TRUE if we are looking at the inside of an object */

float getpower();
float cuberoot();
float sqr();
float getintersection();
float gettorusdistance();
float solvequarticequation();
float checkifinshadow();

float HEIGHT = (WIDTH*GETMAXY/GETMAXX);
int done;
int xstart,ystart,xend,yend;
FILE *f;
int WAXED,TODISK,NUMBOBJECTS,ANTIALIASING,KEEPIMAGE,ZOOM,AMBIANT,NUMLIGHTS;
float WINDOWX1,WINDOWY1,WINDOWX2,WINDOWY2;
int ZOOMX1,ZOOMY1,ZOOMX2,ZOOMY2;
int SURFACE,MODIFIED,GID;
float EYEX,EYEY,EYEZ,CENTERX,CENTERY,CENTERZ; /* View points */
char str[80];  /* All-purpose string */

main(argc,argv)
```

```
int argc;
char *argv[];
{
#ifdef DOGRAPH
  /*if (ismex())
  {
    printf ("%cCannot run this program from MEX.  Quit MEX and try again.\n",7);
    exit(1);
  }*/
#endif
  initallvars();
  if (argc > 1)
  {
    TODISK=TRUE;
    loadthisworld(argv[1]);
    drawpicture();
  }
  else
    while (menu());
}

initallvars()
{
  reseteverything();
  WAXED=FALSE;    /* WAXED means that the surface reflects */
  NUMBOBJECTS=0;  /* Number of objects in this world       */
  TODISK=FALSE;   /* TRUE would send the output directly to disk */
  ANTIALIASING=TRUE;  /* Will anti-alias the final image */
  KEEPIMAGE=FALSE;    /* TRUE would save the image after it is generated */
  ZOOM=0;       /* Used when zooming into parts of the picture */
  SURFACE=CHECKERBOARD;  /* Initial surface */
  AMBIANT=100;    /* Ambiant light.  i.e., light in shadows, etc. Out of 255 */
  MODIFIED=FALSE;  /* TRUE=This world was modified and not saved to disk */
#ifndef DOGRAPH
  TODISK=TRUE;
#endif
}

int menu()
{
  char ch;

  clrscr();
  printf ("\n\n                                        Ray Tracer\n");
  printf ("\n\n                                           by\n");
  printf ("    "                                    Lawrence Kesteloot\n");
  printf ("    "                                          and\n");
  printf ("    "                                     Fredrik Fatemi\n");
  printf ("\n\n");
  printf ("    G - Go ahead and draw the image\n");
  printf ("    D - To disk/to screen (currently to %s)\n",TODISK?"DISK":"SCREEN");
  printf ("    K - Toggle Keep (save image to file '%s' - Currently %s)\n",FILENAME,KEEP
IMAGE?"ON":"OFF");
  printf ("    L - Load a graphic file for display\n");
  printf ("    M - Modify this world\n");
  printf ("    N - Load a New World\n");
  printf ("    P - Parameters (Platform=%d, Ambiant=%d, Eye=%d,%d,%d, Center=%d,%d,%d)\n
",SURFACE,AMBIANT,(int)EYEX,(int)EYEY,(int)EYEZ,(int)CENTERX,(int)CENTERY,(int)CENTERZ);
  printf ("    R - Reset window\n");
  printf ("    S - Save this world\n");
  printf ("    W - Toggle Waxed (currently %s)\n",WAXED?"ON":"OFF");
  printf ("    Q - Quit Ray Tracer\n");
  printf ("\n\n>");
  gets(str);
  ch=toupper(str[0]);
  switch (ch)
  {
    case 'G': sleep(2); drawpicture(); break;  /* SLEEP is necessary */
    case 'D': TODISK=!TODISK; break;
    case 'K': KEEPIMAGE=!KEEPIMAGE; break;
    case 'L': loadfile(); break;
    case 'M': modifyworld(); break;
    case 'N': loadworld(); break;
    case 'P': changeparameters();
    case 'R': resetwindow(); break;
    case 'S': saveworld(); break;
    case 'W': WAXED=!WAXED; MODIFIED=TRUE; break;
    case 'Q': if (MODIFIED)
              {
                printf ("WORLD was modified.  Are you sure you want to quit? [y/N] ");
                gets(str);
                if (toupper(str[0])=='Y') return (FALSE);
              }
              else return (FALSE);
  }
  return(TRUE);
}

drawpicture()
{
  if (!TODISK) initialize(); else opendiskfile();
  drawimage();
  if (!TODISK) cleanup(); else closediskfile();
}

initialize ()
{
#ifdef DOGRAPH
#ifdef IRIS4D
  prefposition (0,1023,0,767);
  foreground();
  GID=winopen ("Ray Tracer");
#else
  ginit();
  cursoff();
#endif
  tpoff();
  RGBmode();
  gconfig();
  RGBcolor(0,0,0);
  clear();
  qdevice (LEFTMOUSE);
  qdevice (MIDDLEMOUSE);
  qdevice (RIGHTMOUSE);
  qdevice (KEYBD);
  qreset();
#endif
}

opendiskfile()
{
  if ((f=fopen(FILENAME,"w"))==NULL)
  {
    printf ("RAY.C:  Unable to open file %s for output.",FILENAME);
    exit(1);
  }
}
```

```
drawimage()
{
  float height,width,x;

  initothervars();
  do
  {
    ZOOM=0;
    drawbobjects();
    if (ZOOM==3)
    {
      width=WINDOWX2-WINDOWX1;
      height=WINDOWY2-WINDOWY1;
      WINDOWY2=(1.0*ZOOMY2/GETMAXY)*height+WINDOWY1;
      WINDOWY1=(1.0*ZOOMY1/GETMAXY)*height+WINDOWY1;
      WINDOWX2=(1.0*ZOOMX2/GETMAXX)*width+WINDOWX1;
      WINDOWX1=(1.0*ZOOMX1/GETMAXX)*width+WINDOWX1;
      x=(WINDOWX1+WINDOWX2)/2.0;
      height=WINDOWY2-WINDOWY1;
      width=height*GETMAXX/GETMAXY;
      WINDOWX1=x-width/2;
      WINDOWX2=x+width/2;
    }
  }
  while (ZOOM==3);
  if ((!done)&&(!TODISK))
  {
    if (ANTIALIASING) antialias();
#ifdef DOGRAPH
    if (KEEPIMAGE) saveimage (FILENAME);
#endif
  }
}

cleanup()
{
#ifdef DOGRAPH
  if (!done)
  {
    ringbell();
    ringbell();
    while (qtest()==0);
  }
  greset();
  color(BLACK);
  clear();
  tpon();
#ifdef IRIS4D
  winclose(GID);
#else
  gexit();
#endif
#endif
}

closediskfile()
{
  fclose(f);
#ifdef DOGRAPH
  ringbell();
#endif
}

initothervars()
{
  /* This routine defines some often-used variables so that they don't */
  /* have to be calculated at every pixel */

  int i;

  for (i=0;i<NUMBOBJECTS;i++)
  {
    if (bobject[i].objecttype==CONEOBJECT)
    {
      bobject[i].sina=sin(bobject[i].A);
      bobject[i].cosa=cos(bobject[i].A);
      bobject[i].tana=tan(bobject[i].A);
    }
  }
  ZOOM=0;
}

reseteverything()
{
  NUMLIGHTS=1;
  bobject[100].x= -300;
  bobject[100].y=500;
  bobject[100].z=400;
  bobject[100].r=100;  /* Radius of bulb */
  bobject[100].reflect=10000;  /* BRIGHTNESS of light */
  bobject[100].objecttype=SPHEREOBJECT;
  resetwindow();
  CENTERX=0;  /* Point you are looking at in the world */
  CENTERY=0;
  CENTERZ= -300;
  EYEX=0;     /* Coordinate of eye */
  EYEY=0;
  EYEZ=150;
  SURFACE=0;  /* Checkerboard */
}

resetwindow()
{
  WINDOWX1= -WIDTH/2;  /* Set ZOOM back to default */
  WINDOWY1= -HEIGHT/2;
  WINDOWX2= WIDTH/2;
  WINDOWY2= HEIGHT/2;
}

drawbobjects()
{
  int xx,yy,x,y,step,R,G,B,val,skip;
  short d;
  float X,Y,height,width;
  RGBvalue RA[1024],GA[1024],BA[1024],downRA[1024],downGA[1024],downBA[1024];

  xstart=0;
  xend=GETMAXX;
  ystart=0;
  yend=GETMAXY;
  done=FALSE;
  skip=FALSE;
  width=WINDOWX2-WINDOWX1;
  height=WINDOWY2-WINDOWY1;

  step=INITSTEP;
  if (TODISK) step=1;
  do
```

```
  {
    for(y=ystart;(y<=yend)&&(!done);y+=step)
    {
      if (TODISK&&(y>ystart))
      {
        memcpy(downRA,RA,1024);
        memcpy(downGA,GA,1024);
        memcpy(downBA,BA,1024);
      }
      Y = WINDOWY1+(height*y/GETMAXY);
      for (x=xstart;x<=xend;x+=step)
      {
        if (((step==INITSTEP)||(x%(step*2)>0)||(y%(step*2)>0))||skip||TODISK)
        {
          X = WINDOWX1+(width*x/GETMAXX);
          getpixelcolor(X,Y,&R,&G,&B);
          if (TODISK)
          {
            RA[x]=(char)R;
            GA[x]=(char)G;
            BA[x]=(char)B;
          }
          else
          {
#ifdef DOGRAPH
            RGBcolor(R,G,B);
            if (step==1)
              pnt2i(x,y);
            else
              rectfi(x,y,x+step-1,y+step-1);
#endif
          }
        }
      }
#ifdef DOGRAPH
      if (!TODISK&&qtest())
      {
        d=1;
        do
        {
          val=qread(&d);
          if (val==KEYBD)
          {
            if (d=='z') ZOOM=1;
          }
          if (d==1) curson();
          if (d==0)
          {
            cursoff();
            xx=getvaluator(MOUSEX);
            yy=getvaluator(MOUSEY);
            if (val==LEFTMOUSE)
            {
              if (ZOOM==1)
              {
                ZOOMX1=xx;
                ZOOMY1=yy;
                ZOOM=2;
              }
              if ((xx<xstart)||(yy<ystart)) skip=TRUE;
              xstart=xx-xx%step;
              ystart=yy-yy%step;
              x=xstart;
            }
            if (val==RIGHTMOUSE)
            {
              if (ZOOM==2)
              {
                ZOOMX2=xx;
                ZOOMY2=yy;
                ZOOM=3;
              }
              if ((xx>xend)||(yy>yend)) skip=TRUE;
              xend=xx-xx%step;
              yend=yy-yy%step;
            }
            if (val==MIDDLEMOUSE) done=TRUE;
          }
        }
        while ((d!=0)&&(val!=KEYBD));
      }
#endif
      if (TODISK)
      {
        if (ANTIALIASING&&(y>ystart))
          antialiasrow(RA,GA,BA,downRA,downGA,downBA,y);
        fwrite (downRA,1,1024,f);
        fwrite (downGA,1,1024,f);
        fwrite (downBA,1,1024,f);
      }
    }
    step/=2;
  }
  while ((step >= ENDSTEP)&&(ZOOM!=3));
  if (TODISK)
  {
    fwrite (RA,1,1024,f);
    fwrite (GA,1,1024,f);
    fwrite (BA,1,1024,f);
  }
}

getpixelcolor(X,Y,RR,GG,BB)
float X,Y;
int *RR,*GG,*BB;
{
  struct RAY ray;
  float r,rr,th,a,XX,YY,ZZ,XXX,YYY,ZZZ;

  /* The next few lines are the transformation to move the window around */

  rr=1.0/sqrt(sqr(EYEX-CENTERX)+sqr(EYEY-CENTERY)+sqr(EYEZ-CENTERZ));
  XX=EYEX-(EYEX-CENTERX)*rr;
  YY=EYEY-(EYEY-CENTERY)*rr;
  ZZ=EYEZ-(EYEZ-CENTERZ)*rr;

  XXX=EYEX-XX;
  YYY=EYEY-YY;
  ZZZ=EYEZ-ZZ;
/*  ray.X1=(X)*sqrt(ZZZ*ZZZ+YYY*YYY)+(-Y)*XXX*YYY-XXX;
  ray.Y1=(Y)*sqrt(ZZZ*ZZZ+XXX*XXX)-YYY;
  ray.Z1=(-X)*XXX+(-Y)*YYY-ZZZ;*/

  rr=sqrt(XXX*XXX+ZZZ*ZZZ);
  ray.X1=(Y*XXX*YYY+X*ZZZ)/rr-XXX;
  ray.Y1=Y*rr-YYY;
  ray.Z1=(-X*XXX+Y*ZZZ*YYY)/rr-ZZZ;
```

```
  ray.X2=EYEX;
  ray.Y2=EYEY;
  ray.Z2=EYEZ;

  *RR=(int)((-Y+HEIGHT/2)/HEIGHT*255);
  if (*RR > 255) *RR=255;
  if (*RR < 0) *RR=0;
  *GG= *RR;
  *BB= 255;

  getraycolor (ray,RR,GG,BB,FALSE);
}

getraycolor (ray,RR,GG,BB,inglass)
struct RAY ray;
int *RR,*GG,*BB,inglass;
{
  /* getraycolor returns the color of the object at the point at which the */
  /* ray intersects the nearest object.  If the object is reflective, then */
  /* getraycolor is called recursively with the reflected or refracted ray */

  int r,g,b,i,j,RRR,GGG,BBB,closest,intersect,isinside;
  float x,y,z,t,A,B,C,q,cx,cy,cz,a,nx,ny,nz,lx,ly,lz,rr,rad,inshadow,temp,mrad,ma;
  struct RAY newray;

  rr=1.0/sqrt(ray.X1*ray.X1+ray.Y1*ray.Y1+ray.Z1*ray.Z1);
  ray.X1*=rr;
  ray.Y1*=rr;
  ray.Z1*=rr;
  ray.X2+=ray.X1;
  ray.Y2+=ray.Y1;
  ray.Z2+=ray.Z1;
  intersect=FALSE;
  q=99999999999;
  for (i=0;i<NUMBOBJECTS;i++) /* Find nearest object */
  {
    t=getintersection(i,ray,inglass);
    if ((t < q) && (t > 0))
    {
      q=t;
      closest=i;
      intersect=TRUE;
    }
  }
  if (intersect)
  {
    t=q;
    i=closest;
    isinside=inside[i];
    {
      x=ray.X1*t+ray.X2;
      y=ray.Y1*t+ray.Y2;
      z=ray.Z1*t+ray.Z2;
      inshadow=checkifinshadow(x,y,z);  /* Returns brightness at that point */
      getnormalvector(i,x,y,z,&nx,&ny,&nz);
      if (bobject[i].Red==SILVER)
      {
        *RR= *GG= *BB=150;
      }
      else
      {
        *RR=bobject[i].Red;
        *GG=bobject[i].Green;
        *BB=bobject[i].Blue;
      }
      rr=1.0/sqrt(nx*nx+ny*ny+nz*nz);
      if (isinside) rr= -rr; /* We're looking at the INSIDE of an object */
      nx*=rr; ny*=rr; nz*=rr;
      a=ma=0;
      for (j=100;j<(100+NUMLIGHTS);j++)
      {
        lx=bobject[j].x-x;
        ly=bobject[j].y-y;
        lz=bobject[j].z-z;
        temp=1.0/sqrt(lx*lx+ly*ly+lz*lz);
        lx*=temp; ly*=temp; lz*=temp;
        rad=lx*nx+ly*ny+lz*nz;
        if (bobject[i].reflect==MATTE)
        {
          getreflection(ray.X1,ray.Y1,ray.Z1,nx,ny,nz,&newray.X1,&newray.Y1,&newray.Z1);
          mrad=newray.X1*lx+newray.Y1*ly+newray.Z1*lz;
          ma+=mrad;
        }
        if (rad > 0) a+=rad;
      }
      a/=NUMLIGHTS;
      if (a<0.0) a=0.0;
      if (a>1.0) a=1.0;
      *RR=(int)(*RR*a*inshadow);
      *GG=(int)(*GG*a*inshadow);
      *BB=(int)(*BB*a*inshadow);
      if (bobject[i].reflect==MATTE)
      {
        ma/=NUMLIGHTS;
        if (ma<0.0) ma=0.0;
        if (ma>1.0) ma=1.0;
        ma*=ma*ma;
        *RR=(int)(*RR+(255-*RR)*ma);
        *GG=(int)(*GG+(255-*GG)*ma);
        *BB=(int)(*BB+(255-*BB)*ma);
      }
      if ((bobject[i].reflect != DULL)&&(bobject[i].reflect != MATTE))
      {
        getreflection(ray.X1,ray.Y1,ray.Z1,nx,ny,nz,&newray.X1,&newray.Y1,&newray.Z1);
        newray.X2=x;
        newray.Y2=y;
        newray.Z2=z;
        if (bobject[i].Red==SILVER)
        {
          RRR= *RR;
          GGG= *GG;
          BBB= *BB;
          getraycolor(newray,&RRR,&GGG,&BBB,inglass);
          *RR=(RRR+*RR)*0.5;
          *GG=(GGG+*GG)*0.5;
          *BB=(BBB+*BB)*0.5;
          if (*RR > 255) *RR=255;
          if (*GG > 255) *GG=255;
          if (*BB > 255) *BB=255;
        }
        else
        if (bobject[i].reflect==GLASS)
        {
          if (!inglass)
            getraycolor(newray,RR,GG,BB,inglass);
          getrefraction(ray.X1,ray.Y1,ray.Z1,nx,ny,nz,&newray.X1,&newray.Y1,&newray.Z1,&i
nglass);
          if (inglass)
```

```
          {
            RRR= *RR;
            GGG= *GG;
            BBB= *BB;
            getraycolor(newray,&RRR,&GGG,&BBB,inglass);
            *RR=(3*RRR+*RR)/4;
            *GG=(3*GGG+*GG)/4;
            *BB=(3*BBB+*BB)/4;
            if (*RR > 255) *RR=255;
            if (*GG > 255) *GG=255;
            if (*BB > 255) *BB=255;
          }
          else
          {
            getraycolor(newray,RR,GG,BB,inglass);
            if (*RR > 255) *RR=255;
            if (*GG > 255) *GG=255;
            if (*BB > 255) *BB=255;
          }
        }
        else
        {
          RRR= *RR;
          GGG= *GG;
          BBB= *BB;
          getraycolor(newray,&RRR,&GGG,&BBB,inglass);
          *RR=(5*RRR+*RR)/6;
          *GG=(5*GGG+*GG)/6;
          *BB=(5*BBB+*BB)/6;
          if (*RR > 255) *RR=255;
          if (*GG > 255) *GG=255;
          if (*BB > 255) *BB=255;
        }
      }
    }
  }
  cy= -50.0;  /* Y position of surface */
  if (ray.Y1 != 0)
  {
    t=(cy-ray.Y2)/ray.Y1;
    if ((t > 0) && (t < q))
    {
      q=t;
      cx=ray.X1*t+ray.X2;
      cz=ray.Z1*t+ray.Z2;
      if ((cz < 0) && (cz > -400) && (fabs(cx) < 120))
      {
        intersect=TRUE;
        inshadow=checkifinshadow(cx,cy,cz);
        t=0;
        for (i=100;i<(100+NUMLIGHTS);i++)
        {
          q=(bobject[i].y-cy)/sqrt(sqr(bobject[i].x-cx)+sqr(bobject[i].y-cy)+sqr(bobject
[i].z-cz));
          if (bobject[i].reflect > 5000) q=q/2+0.5;
          t+=q;
        }
        inshadow*=t/NUMLIGHTS;
        if (WAXED)
        {
          if (SURFACE==WATER)
          {
            rad=sqrt(cx*cx+cz*cz);
            newray.Y1=1.0/(1.0+RIPPLE*RIPPLE*AMP*AMP*sqr(cos(RIPPLE*rad)));
            newray.X1=cx*sqrt(1.0-newray.Y1*newray.Y1)/rad;
            newray.Z1=cz*sqrt(1.0-newray.Y1*newray.Y1)/rad;
            if (cos(rad) > 0)
            {
              newray.X1= -newray.X1;
              newray.Z1= -newray.Z1;
            }
          }
          else
          {
            newray.X1=ray.X1;
            newray.Y1= -ray.Y1;
            newray.Z1=ray.Z1;
          }
          newray.X2=cx;
          newray.Y2=cy;
          newray.Z2=cz;
          RRR=GGG=BBB=0;
          getraycolor(newray,&RRR,&GGG,&BBB,inglass);
        }
        if (SURFACE==CHECKERBOARD)
        {
          cx=(cx+150)/20.0;
          cz/=20.0;
          if (((int)(cx) + (int)(cz)) % 2)
          {
            *RR=255; *GG=0; *BB=0;
          }
          else
          {
            *RR=255; *GG=255; *BB=255;
          }
        }
        if (SURFACE==MANDELBROT)
        {
          cx=(pmax-pmin)*(cx+100.0)/200.0+pmin;
          cz=(qmax-qmin)*(cz+400.0)/400.0+qmin;
          getmandelbrot(cx,cz,RR,GG,BB);
        }
        if (SURFACE==WATER)  /* Not working yet */
        {
          a=0.0;
          for (j=100;j<(100+NUMLIGHTS);j++)
          {
            a+=((bobject[j].x-cx)*newray.X1+(bobject[j].y-cy)*newray.Y1+(bobject[j].z-cz)
*newray.Z1)/(sqrt(sqr(bobject[j].x-cx) + sqr(bobject[j].y-cy) + sqr(bobject[j].z-cz)));
          }
          a/=NUMLIGHTS;
          if (a<0.0) a=0.0;
          if (a>1.0) a=1.0;
          *GG= *RR= *BB=(int)(255*a);
          printf ("%f %f %f %f\n",a,newray.X1,newray.Y1,newray.Z1);
        }
        *RR=(int)(*RR*inshadow);
        *GG=(int)(*GG*inshadow);
        *BB=(int)(*BB*inshadow);
        if (WAXED)
        {
          *RR=(4**RR+RRR)/5;
          *GG=(4**GG+GGG)/5;
          *BB=(4**BB+BBB)/5;
          if (*RR > 255) *RR=255;
          if (*GG > 255) *GG=255;
          if (*BB > 255) *BB=255;
```

```
        }
      }
    }
  }
  for (j=100;j<(100+NUMLIGHTS);j++) /* This checks to see if the ray */
  {                                 /* intersects any light bulbs    */
    t=getintersection(j,ray,FALSE);   /*  Light  */
    if (t > 0)
    {
      inshadow=0;
      for (i=0;i<NUMBOBJECTS;i++)
      {
        t=getintersection(i,ray,FALSE);
        if (t > 0) inshadow=1;
      }
      if (inshadow==0)
      {
        *RR=1000;  /* 1000 out of 255 means the bulb blooms. This is done */
        *GG=1000;  /* so that if it is averaged with any other color, it  */
        *BB=200;   /* will still look bright white. BB=200= it is yellow */
        intersect=TRUE;
      }
    }
  }
  return;
}

getnormalvector (i,x,y,z,nx,ny,nz)
int i;
float x,y,z,*nx,*ny,*nz;
{
  /* Returns the normal vector to the surface of object "i" at point x,y,z */

  float t;

  *nx=x-bobject[i].x;
  *ny=y-bobject[i].y;
  *nz=z-bobject[i].z;
  switch (bobject[i].objecttype)
  {
    case SPHEREOBJECT:
      break;
    case TORUSOBJECT:
      t=sqrt(*nx**nx+*ny**ny);
      *nx-=(bobject[i].A**nx)/t;
      *ny-=(bobject[i].A**ny)/t;
      break;
    case ELLIPSELOIDOBJECT:
      *nx*=bobject[i].C*bobject[i].B/bobject[i].A;
      *ny*=bobject[i].A*bobject[i].C/bobject[i].B;
      *nz*=bobject[i].A*bobject[i].B/bobject[i].C;
      break;
    case CONEOBJECT:
      *nx*= -bobject[i].cosa/(*ny*bobject[i].tana);
      *nz*= -bobject[i].cosa/(*ny*bobject[i].tana);
      *ny= bobject[i].sina;
      break;
    case HYPERBOLOID:
      *ny*= -(bobject[i].A*bobject[i].A);
      break;
    case CYLINDER:
      *ny=0;
      break;
  }
}

float getintersection (i,ray,inglass)
int i;
struct RAY ray;
int inglass;
{
  /* The program spends 33% of its time in this one procedure. Amazing.  */
  /* Anyway, it checks to see if the ray "ray" intersects the object "i" */
  /* and if so, it will return the distance to that object in terms of   */
  /* lengths of "ray".  */

  float d,t,tt,A,B,C,AA,BB,CC,X1,Y1,Z1,X2,Y2,Z2,tangent;

  X1=ray.X1;
  Y1=ray.Y1;
  Z1=ray.Z1;
  X2=ray.X2-bobject[i].x;
  Y2=ray.Y2-bobject[i].y;
  Z2=ray.Z2-bobject[i].z;
  inside[i]=FALSE;

  {
    switch (bobject[i].objecttype)
    {
      case ELLIPSELOIDOBJECT:
        AA=1.0/(bobject[i].A*bobject[i].A);
        BB=1.0/(bobject[i].B*bobject[i].B);
        CC=1.0/(bobject[i].C*bobject[i].C);
        A=X1*X1*AA+Y1*Y1*BB+Z1*Z1*CC;
        B=2*(X1*X2*AA+Y1*Y2*BB+Z1*Z2*CC);
        C=X2*X2*AA+Y2*Y2*BB+Z2*Z2*CC-1;
        break;
      case SPHEREOBJECT:
      case TORUSOBJECT:
        A=X1*X1+Y1*Y1+Z1*Z1;
        B=2*(X1*X2+Y1*Y2+Z1*Z2);
        C=X2*X2+Y2*Y2+Z2*Z2-bobject[i].r*bobject[i].r;
        break;
      case CONEOBJECT:
        tangent=sqr(bobject[i].tana);
        A=X1*X1+Z1*Z1-Y1*Y1*tangent;
        B=2*(X1*X2+Z1*Z2-Y1*Y2*tangent);
        C=X2*X2+Z2*Z2-Y2*Y2*tangent;
        break;
      case HYPERBOLOID:
        AA=sqr(bobject[i].A);
        BB=sqr(bobject[i].B);
        A=X1*X1+Z1*Z1-AA*Y1*Y1;
        B=2*(X1*X2+Z1*Z2-AA*Y1*Y2);
        C=X2*X2+Z2*Z2-AA*Y2*Y2-BB;
        break;
      case CYLINDER:
        A=X1*X1+Z1*Z1;
        B=2*(X1*X2+Z1*Z2);
        C=X2*X2+Z2*Z2-bobject[i].r*bobject[i].r;
        break;
    }
    d=B*B-4*A*C;
    if (d > 0)
    {
      if (bobject[i].objecttype==TORUSOBJECT)
      {
        t=(-B-sqrt(d))/(A+A);
```

```
        ray.X2=ray.X1*t+ray.X2;
        ray.Y2=ray.Y1*t+ray.Y2;
        ray.Z2=ray.Z1*t+ray.Z2;
        tt=gettorusdistance (i,ray);
        if (tt < 0) t=tt; else t+=tt;
      }
      else
      {
        if (inglass)
          t=(-B+sqrt(d))/(A+A);
        else
          t=(-B-sqrt(d))/(A+A);
        if ((bobject[i].top != 0)||(bobject[i].bottom != 0))
        {
          if ((Y1*t+Y2 < bobject[i].bottom)||(Y1*t+Y2 > bobject[i].top)||(t<0))
            if (!inglass)
            {
              t=(-B+sqrt(d))/(A+A);
              if ((Y1*t+Y2 < bobject[i].bottom)||(Y1*t+Y2 > bobject[i].top))
                t= -1.0;
              else
                inside[i]=TRUE;
            }
        }
      }
    }
    else
      t=d;
  }
  return (t);
}

float gettorusdistance (i,ray)
int i;
struct RAY ray;
{
  /* This will return the distance, as in GetIntersection, from the */
  /* current ray to its intersection with the torus of object "i",  */
  /* in terms of lengths of ray "ray". Since a ray can intersect    */
  /* a torus in four places (think about it), the equation is a     */
  /* quartic equation, and the routine solvequacticequation is used */
  /* to find the lowest positive root. Since finding the roots of   */
  /* a quartic equation is real math-intensive, round-off problems  */
  /* sometimes cause messy toruses (or tori?)                       */

  float A,B,C,D,t,AA,BB,X1,X2,Y1,Y2,Z1,Z2,PartA,PartAAA,H;

  X1=ray.X1; X2=ray.X2-bobject[i].x;
  Y1=ray.Y1; Y2=ray.Y2-bobject[i].y;
  Z1=ray.Z1; Z2=ray.Z2-bobject[i].z;
  AA=bobject[i].A;
  BB=bobject[i].B;

  PartA=4.0*(X1*X2+Y1*Y2+Z1*Z2);
  PartAAA=1.0*(X2*X2+Y2*Y2+Z2*Z2+AA*AA-BB*BB);
  H=4.0*AA*AA;
  A=PartA;
  B=0.25*PartA*PartA+2*PartAAA-H*(X1*X1+Y1*Y1);
  C=PartA*PartAAA-2*H*(X1*X2+Y1*Y2);
  D=PartAAA*PartAAA-H*(X2*X2+Y2*Y2);

  t=solvequarticequation (A,B,C,D);

  return (t);
}

float cuberoot (x)
float x;
{
  /* Returns the cube root of X */

  if (x==0) return (0.0);
  if (x < 0)
    return (-exp(log(-x)/3));
  return (exp(log(x)/3));
}

float solvequarticequation(A,B,C,D)
float A,B,C,D;
{
  /* Don't even try to figure out what's going on in this function */

  float part1,cosPhi,TanPhi,root1,disc,LL,KK,GGG,asqd,part2,P,Q,R,G,HH,PP;
  float DDDD,H,Phi,RR,SS,R1,R2,R3,R4,t;

  DDDD=4.0*D;
  asqd=A*A;
  P= -B;
  Q=A*C-DDDD;
  R= -asqd*D+B*DDDD-C*C;
  PP=B*B;
  G=(3*Q-PP)/3;
  GGG=G*G*G;
  H=P*(2*PP-9*Q)/27+R;
  HH=H/2;
  disc=H*H/4+GGG/27;
  if (disc>=0)
  {
    disc=sqrt(disc);
    root1=cuberoot(-HH+disc)-cuberoot(HH+disc);
  }
  if (disc<0)
  {
    cosPhi= -HH*sqrt(-27/GGG);
    TanPhi=sqrt(1-cosPhi*cosPhi)/cosPhi;
    Phi=atan(TanPhi);
    root1=2*sqrt(-G/3)*cos(Phi/3);
  }
  root1-=P/3;
  RR=asqd/4+P+root1;
  if (RR<0)
    return (-1.0);
  R1=R2=R3=R4= -1.0;
  SS=sqrt(RR);
  if (fabs(SS) < 0.00001)
  {
    part1=3*A*A/4-2*B;
    part2=2*sqrt(root1*root1-4*D);
  }
  else
  {
    part1=3*asqd/4-RR+2*P;
    part2=(4*A*B-8*C-asqd*A)/(4*SS);
  }
  KK=part1+part2;
  if (KK>=0)
  {
    KK=sqrt(KK);
```

```
    R1= -A/4+SS/2+KK/2;
    R2= -A/4+SS/2-KK/2;
  }
  LL=part1-part2;
  if (LL>=0)
  {
    LL=sqrt(LL);
    R3= -A/4-SS/2+LL/2;
    R4= -A/4-SS/2-LL/2;
  }
  t=40000;
  if ((R1 > 0) && (R1 < t)) t=R1;
  if ((R2 > 0) && (R2 < t)) t=R2;
  if ((R3 > 0) && (R3 < t)) t=R3;
  if ((R4 > 0) && (R4 < t)) t=R4;
  if (t==40000)
    return (-2.0);
  return (t);
}

float checkifinshadow(x,y,z)
float x,y,z;
{
  /* This function will return a number from 0 to 1, indicating the */
  /* brightness of the point x,y,z, according to which lights it is */
  /* being lit by, and its distance to those lights.                */

  int i,j,intersect;
  float A,B,C,t,tt,rr;
  struct RAY ray;

  tt=0.0;
  for (j=100;j<(100+NUMLIGHTS);j++)
  {
    ray.X1=bobject[j].x-x;
    ray.Y1=bobject[j].y-y;
    ray.Z1=bobject[j].z-z;
    rr=1.0/sqrt(ray.X1*ray.X1+ray.Y1*ray.Y1+ray.Z1*ray.Z1);
    ray.X2=x+ray.X1*rr;
    ray.Y2=y+ray.Y1*rr;
    ray.Z2=z+ray.Z1*rr;
    intersect=FALSE;
    for (i=0;i<NUMBOBJECTS;i++)
    {
      t=getintersection(i,ray,FALSE);
      if ((t > 0)&&(t < 1)) intersect=TRUE;
    }
    if (!intersect)
    {
      rr*=rr;   /* Square rr to get square of distance */
      rr*=1.0*bobject[j].reflect*bobject[j].reflect;
      if (rr > 1.0) rr=1.0;
      tt+=rr;
    }
  }
  tt/=NUMLIGHTS;
  tt=1.0*AMBIANT/255+tt*(255.0-AMBIANT)/255.0;
  return(tt);
}

getreflection (A1,B1,C1,A2,B2,C2,A3,B3,C3)
float A1,B1,C1,A2,B2,C2,*A3,*B3,*C3;
{
  /* Given incoming vector A1,B1,C1 and normal vector A2,B2,C2, it will */
  /* return reflected vector A3,B3,C3 */
  float t;

  t=1.0/sqrt(A1*A1+B1*B1+C1*C1);
  A1*=t; B1*=t; C1*=t;
  t=1.0/sqrt(A2*A2+B2*B2+C2*C2);
  A2*=t; B2*=t; C2*=t;
  t=2.0*(A1*A2+B1*B2+C1*C2);
  *A3=A1-t*A2;
  *B3=B1-t*B2;
  *C3=C1-t*C2;
}

getrefraction (I1,I2,I3,N1,N2,N3,R1,R2,R3,inglass)
float I1,I2,I3,N1,N2,N3,*R1,*R2,*R3;
int *inglass;
{
  /* Given incoming vector I1,I2,I3, and normal vector N1,N2,N3, it will */
  /* return the refracted vector R1,R2,R3.  If "inglass" is false, then  */
  /* it will refract according to the ratio from air to glass, otherwise */
  /* it will refract according to the ratio from glass to air.           */

  float t,a,th,n1=1.0,n2=1.52,m,o;

  t=1.0/sqrt(sqr(I1)+sqr(I2)+sqr(I3));
  I1*=t; I2*=t; I3*=t;
  t=1.0/sqrt(sqr(N1)+sqr(N2)+sqr(N3));
  if (!*inglass) t= -t;
  N1*=t; N2*=t; N3*=t;
  th=acos(I1*N1+I2*N2+I3*N3);
  if (*inglass)
    m=sin(th)*n2/n1;
  else
    m=sin(th)*n1/n2;
  a=asin(m);

  *inglass= !*inglass;
  o=sin(th-a);
  *R1=I1*m+o*N1;
  *R2=I2*m+o*N2;
  *R3=I3*m+o*N3;
}

float getpower(x,y)
float x;
int y;
{
  /* Returns x^y */

  float r;

  r = 1.0;
  do
  {
    r*=x;
  }
  while (--y > 0);
  return (r);
}

float sqr(x)
float x;
{
  /* This seems simple enough */
```

```
  return (x*x);
}

antialias()
{
  int y;
  RGBvalue red[1024],green[1024],blue[1024],downred[1024],downgreen[1024],downblue[1024]
;

#ifdef DOGRAPH
  ringbell();
  for (y=ystart+1;y<=yend;y++)
  {
    cmov2i(0,y);
    readRGB (1024,red,green,blue);
    cmov2i(0,y-1);
    readRGB (1024,downred,downgreen,downblue);
    antialiasrow(red,green,blue,downred,downgreen,downblue,y);
    cmov2i(0,y-1);
    writeRGB (1024,downred,downgreen,downblue);
  }
#endif
}

antialiasrow(red,green,blue,downred,downgreen,downblue,y)
RGBvalue red[],green[],blue[],downred[],downgreen[],downblue[];
int y;
{
  /* Will anti-alias a row of graphics by checking to see if the neighboring */
  /* pixels are different.  If they are, then the current pixel is assumed   */
  /* to be on a border, and is broken down into 16 sub-pixels and the colors */
  /* are averaged.  This creates an illusion and edges look smoother.        */

  int x,R,G,B,r,g,b;
  float xx,yy,X,Y,width,height;
  RGBvalue newred[1024],newgreen[1024],newblue[1024];

  y--;
  width=WINDOWX2-WINDOWX1;
  height=WINDOWY2-WINDOWY1;
  memcpy (newred,downred,1024);
  memcpy (newgreen,downgreen,1024);
  memcpy (newblue,downblue,1024);
  for (x=xstart;x<xend;x++)
    if (diff(red,downred,x) || diff(green,downgreen,x) || diff(blue,downblue,x))
    {
      r=g=b=0;
      for (xx=0.0;xx<1.0;xx+=0.25)
      {
        X = WINDOWX1+(width*(x+xx)/GETMAXX);
        for (yy=0.0;yy<1.0;yy+=0.25)
        {
          Y = WINDOWY1+(height*(y+yy)/GETMAXY);
          getpixelcolor(X,Y,&R,&G,&B);
          r+=R; g+=G; b+=B;
        }
      }
      r/=16; g/=16; b/=16;
      newred[x]=r;
      newgreen[x]=g;
      newblue[x]=b;
    }
  memcpy(downred,newred,1024);
  memcpy(downgreen,newgreen,1024);
  memcpy(downblue,newblue,1024);
}

int diff(c,downc,x)
RGBvalue c[],downc[];
int x;
{
  /* Checks to see difference among four pixels */

  int d=10;

  if ((abs(c[x]-downc[x]) > d) || (abs(c[x]-c[x+1]) > d) ||
      (abs(c[x+1]-downc[x+1]) > d) || (abs(downc[x]-downc[x+1]) > d) ||
      (abs(c[x]-downc[x+1]) > d) || (abs(downc[x]-c[x+1]) > d))
    return(TRUE);
  else
    return(FALSE);
}

loadfile()
{
  char a[80];

  clrscr();
  system ("ls -l *.SCR");
  printf ("\n\nName of file to load: ");

  gets(str);
  if (str=="") return;
  strcpy (a,"disprgb ");
  system (strcat (a,str));
}

modifyworld()
{
  /* This function allows the user to modify the current world.  It is  */
  /* real messy cause I wrote it two days ago. Just assume it works and */
  /* skip it, trust me. */

  int i,j,val,done,xx,yy,f,HANDINPUT,MGID;
  short d;
  char ch,s[130];

#ifdef IRIS4D
  prefposition (0,1023,0,767);
  foreground();
  MGID=winopen ("Ray Tracer");
#else
  ginit();
#endif

  tpoff();
  qdevice (LEFTMOUSE);
  qdevice (RIGHTMOUSE);
  qreset();
  done=FALSE;
  HANDINPUT=FALSE;
  do
  {
    color (BLACK);
    clear();
    color (WHITE);
    cmov2i(20,750);
```

```
    charstr ("Current World:");
    if (NUMBOBJECTS==0)
    {
      cmov2i(20,730);
      charstr ("No objects defined.");
    }
    else
    {
      cmov2i(20,730);
      charstr (" # Object          X    Y    Z   R Red Grn Blu Reflect    A    B    C  To
p Bttm");
      cmov2i(20,710);
      charstr ("-- ------       ---- ---- ---- --- --- --- --- -------- ---- ---- ---- ---
- ----");
      for (i=0;i<NUMBOBJECTS;i++)
      {
        cmov2i(20,680-16*i);
        sprintf (s,"%2d %-12s%4d%5d%5d%4d%4d%4d%4d%4d %-8s%4d%5d%5d%5d%5d",i,OBJECTNAME[bob
ject[i].objecttype],(int)bobject[i].x,(int)bobject[i].y,(int)bobject[i].z,(int)bobject[i
].r,(int)bobject[i].Red,(int)bobject[i].Green,(int)bobject[i].Blue,SURFACENAME[bobject[i
].reflect],(int)bobject[i].A,(int)bobject[i].B,(int)bobject[i].C,(int)bobject[i].top,(in
t)bobject[i].bottom);
       charstr (s);
      }
    }
    cmov2i(20,60);
    charstr ("  Click on action:   <ADD OBJECT>  <DELETE OBJECT>  <HAND-INPUT>  <QUIT>")
;
    curson();
    do
    {
      val=qread(&d);
    }
    while (d==0);
    cursoff();
    xx=getvaluator (MOUSEX);
    yy=getvaluator (MOUSEY);
    if (yy < 100)
    {
      if ((xx > 215) && (xx < 311))
      {
        NUMBOBJECTS++;
        i=NUMBOBJECTS-1;
        bobject[i].x=bobject[i].y=bobject[i].z=bobject[i].r=bobject[i].Red=bobject[i].Gr
een=bobject[i].Blue=bobject[i].A=bobject[i].B=bobject[i].C=bobject[i].top=bobject[i].bot
tom=0;
        bobject[i].objecttype=SPHEREOBJECT;
        bobject[i].reflect=SHINE;
        MODIFIED=TRUE;
      }
      if ((xx > 338) && (xx < 458))
      {
        cmov2i(20,100);
        charstr ("Click on the object to delete, or <Right Button> to quit.");
        curson();
        do
        {
          val=qread(&d);
        }
        while (d==0);
        cursoff();
        if (val == LEFTMOUSE)
        {
          yy=getvaluator(MOUSEY);
          if (yy < 696)
          {
            i=(696-yy)/16;
            if ((i >= 0) && (i < NUMBOBJECTS))
            {
              NUMBOBJECTS--;
              for (j=i;j<NUMBOBJECTS;j++)
                bobject[j]=bobject[j+1];
            }
            MODIFIED=TRUE;
          }
        }
      }
      if ((xx > 491) && (xx < 590))
        HANDINPUT=TRUE;
      if ((xx > 612) && (xx < 662)) done=TRUE;
    }
    else
    {
      if (yy < 696)
      {
        if (val==RIGHTMOUSE) HANDINPUT=TRUE;
        i=(696-yy)/16;
        f=0;
        if (xx < 726) f=14;
        if (xx < 680) f=13;
        if (xx < 637) f=12;
        if (xx < 591) f=11;
        if (xx < 548) f=10;
        if (xx < 502) f=9;
        if (xx < 429) f=8;
        if (xx < 392) f=7;
        if (xx < 356) f=6;
        if (xx < 320) f=5;
        if (xx < 286) f=4;
        if (xx < 236) f=3;
        if (xx < 194) f=2;
        if (xx < 145) f=1;
        if (xx < 47) f=0;
        if ((f > 0)&&(i >= 0)&&(i < NUMBOBJECTS))
        {
          if (HANDINPUT)
          {
            handinputfield (i,f);
            HANDINPUT=FALSE;
          }
          else
            editfield (i,f);
          MODIFIED=TRUE;
        }
      }
    }
  }
  while (!done);
  color (BLACK);
  clear();
  greset();
  tpon();
#ifdef IRIS4D
  winclose (MGID);
#else
  gexit();
#endif
}
```

```
editfield(i,f)
int i,f;
{
  int xx,yy,oldyy,orgyy,val,vi,ft,orgi,dy;
  short d;
  float vf,orgf;
  char s[100],ss[100];

  switch (f)
  {
    case 1: ft=1; vi=bobject[i].objecttype; xx=3; break;
    case 2: ft=2; vf=bobject[i].x; xx=15; break;
    case 3: ft=2; vf=bobject[i].y; xx=20; break;
    case 4: ft=2; vf=bobject[i].z; xx=25; break;
    case 5: ft=2; vf=bobject[i].r; xx=29; break;
    case 6: ft=1; vi=bobject[i].Red; xx=33; break;
    case 7: ft=1; vi=bobject[i].Green; xx=37; break;
    case 8: ft=1; vi=bobject[i].Blue; xx=41; break;
    case 9: ft=1; vi=bobject[i].reflect; xx=46; break;
    case 10: ft=2; vf=bobject[i].A; xx=54; break;
    case 11: ft=2; vf=bobject[i].B; xx=59; break;
    case 12: ft=2; vf=bobject[i].C; xx=64; break;
    case 13: ft=2; vf=bobject[i].top; xx=69; break;
    case 14: ft=2; vf=bobject[i].bottom; xx=74; break;
  }
  oldyy=getvaluator(MOUSEY);
  if (ft==2) vi=vf;
  orgi=vi;
  orgf=vf;
  dy=0;
  cursoff();
  do
  {
    switch (f)
    {
      case 1: sprintf (s,"%-12s",OBJECTNAME[vi]); break;
      case 9: sprintf (s,"%-8s",SURFACENAME[vi]); break;
      default: sprintf (s,"%4d",vi); break;
    }
    cmov2i (20+9*xx,680-16*i);
    color (YELLOW);
    charstr (s);
    if (ft==2)
    {
      sprintf (ss,"Real value: %.1f",vf);
      cmov2i (20,100);
      charstr (ss);
    }
    curson();
    do
    {
      yy=getvaluator(MOUSEY);
    }
    while ((oldyy==yy)&&(qtest()==0));
    cursoff();
    if (oldyy != yy)
    {
      cmov2i (20+9*xx,680-16*i);
      color (BLACK);
      charstr (s);
      if (ft==2)
      {
        cmov2i (20,100);
        charstr (ss);
      }
      dy+=yy-oldyy;
      vi=orgi+dy/10;
      if (ft==2) vf=orgf+dy/10.0;
      if (f==1)
      {
        if (vi > 7) vi%=8;
        if (vi < 0) vi= 7-(-(vi+1)%8);
      }
      if (f==9)
      {
        if (vi > 3) vi%=4;
        if (vi < 0) vi= 3-(-(vi+1)%4);
      }
      if ((f >= 6) && (f <= 8))
      {
        if (vi > 255) vi%=8;
        if (vi < 0) vi= 255-(-(vi+1)%256);
      }
      if (f==5)
      {
        if (vi < 0) vi=0;
        if (vf < 0) vf=0;
      }
      if (yy < 100) { yy=760; setvaluator (MOUSEY,760,0,767); }
      if (yy > 760) { yy=100; setvaluator (MOUSEY,100,0,767); }
      oldyy=yy;
      MODIFIED=TRUE;
    }
  }
  while (qtest()==0);
  cursoff();
  switch (f)
  {
    case 1: bobject[i].objecttype=vi; break;
    case 2: bobject[i].x=vf; break;
    case 3: bobject[i].y=vf; break;
    case 4: bobject[i].z=vf; break;
    case 5: bobject[i].r=vf; break;
    case 6: bobject[i].Red=vi; break;
    case 7: bobject[i].Green=vi; break;
    case 8: bobject[i].Blue=vi; break;
    case 9: bobject[i].reflect=vi; break;
    case 10: bobject[i].A=vf; break;
    case 11: bobject[i].B=vf; break;
    case 12: bobject[i].C=vf; break;
    case 13: bobject[i].top=vf; break;
    case 14: bobject[i].bottom=vf; break;
  }
}

handinputfield (i,f)
int i,f;
{
  char s[100],c;
  short d;
  int t,x,val;
  float ff;

  qdevice (KEYBD);
  switch (f)
  {
    case 2: sprintf (s,"Current value of X: %f",bobject[i].x); break;
```

```
    case 3: sprintf (s,"Current value of Y: %f",bobject[i].y); break;
    case 4: sprintf (s,"Current value of Z: %f",bobject[i].z); break;
    case 5: sprintf (s,"Current value of R: %f",bobject[i].r); break;
    case 10: sprintf (s,"Current value of A: %f",bobject[i].A); break;
    case 11: sprintf (s,"Current value of B: %f",bobject[i].B); break;
    case 12: sprintf (s,"Current value of C: %f",bobject[i].C); break;
    case 13: sprintf (s,"Current value of top: %f",bobject[i].top); break;
    case 14: sprintf (s,"Current value of bottom: %f",bobject[i].bottom); break;
    default: return;  /* No point in hand-entering an integer */
  }
  color (YELLOW);
  cmov2i (20,120);
  charstr (s);
  cmov2i (20,100);
  charstr ("Enter new value, or simply <ENTER> to keep as it is: ");
  t=0;
  do
  {
    val=qread(&d);
    if (val == KEYBD)
    {
      c=d;
      if (((c >= '0')&&(c <= '9'))||(c=='.')||(c=='-'))
      {
        s[t++]=c;
        s[t]=0;
        cmov2i(53*9+20,100);
        charstr(s);
      }
      if ((c == 8)&&(t > 0))
      {
        color (BLACK);
        cmov2i(53*9+20,100);
        charstr(s);
        s[--t]=0;
        color (YELLOW);
        cmov2i(53*9+20,100);
        charstr(s);
      }
    }
  }
  while (d != 13);
  if (t > 0)
  {
    ff=atof(s);
    switch (f)
    {
      case 2: bobject[i].x=ff; break;
      case 3: bobject[i].y=ff; break;
      case 4: bobject[i].z=ff; break;
      case 5: bobject[i].r=ff; break;
      case 10: bobject[i].A=ff; break;
      case 11: bobject[i].B=ff; break;
      case 12: bobject[i].C=ff; break;
      case 13: bobject[i].top=ff; break;
      case 14: bobject[i].bottom=ff; break;
    }
  }
  unqdevice (KEYBD);
}

loadworld()
{
  clrscr();
  system ("ls *.world");
  printf ("\nName of the file from which to load a world: ");
  gets(str);
  if (str[0]==0) return;
  loadthisworld(str);
  MODIFIED=FALSE;
}

loadthisworld(str)
char *str;
{
  int i,rr,gg,bb;
  FILE *f;

  if ((f=fopen(str,"r"))==NULL)
  {
    printf ("File '%s' not found.  Strike <RETURN> to continue.",str);
    gets(str);
    return;
  }
  fscanf (f,"%d %d\n",&NUMBOBJECTS,&WAXED);
  fscanf (f,"%f %f %f %f %f %f %d %d %f %f %f %f\n",&EYEX,&EYEY,&EYEZ,&CENTERX,&CENTERY,&
CENTERZ,&SURFACE,&AMBIANT,&WINDOWX1,&WINDOWY1,&WINDOWX2,&WINDOWY2);
  for (i=0;i<NUMBOBJECTS;i++)
  {
    fscanf (f,"%f %f %f %f %d %d %d %d %d %f %f %f %f %f %f %f %f\n",&bobject[i].x,&bobje
ct[i].y,&bobject[i].z,&bobject[i].r,&rr,&gg,&bb,&bobject[i].reflect,&bobject[i].objecttyp
e,&bobject[i].A,&bobject[i].B,&bobject[i].C,&bobject[i].top,&bobject[i].bottom,&bobject[i
].ax,&bobject[i].ay,&bobject[i].az);
    bobject[i].Red=rr;
    bobject[i].Green=gg;
    bobject[i].Blue=bb;
  }
  fscanf (f,"%d\n",&NUMLIGHTS);
  for (i=100;i<(100+NUMLIGHTS);i++)
    fscanf (f,"%f %f %f %f %d\n",&bobject[i].x,&bobject[i].y,&bobject[i].z,&bobject[i].r,
&bobject[i].reflect);
  fclose(f);
}

saveworld()
{
  FILE *f;
  int i;

  clrscr();
  printf ("Name of the file to which to save this world: ");
  gets(str);
  if (str[0]==0) return;
  if ((f=fopen(str,"w"))==NULL)
  {
    printf ("Unable to open file for output . . .  Strike <RETURN> to continue.");
    gets(str);
    return;
  }
  fprintf (f,"%d %d\n",NUMBOBJECTS,WAXED);
  fprintf (f,"%f %f %f %f %f %f %d %d %f %f %f %f\n",EYEX,EYEY,EYEZ,CENTERX,CENTERY,CENTE
RZ,SURFACE,AMBIANT,WINDOWX1,WINDOWY1,WINDOWX2,WINDOWY2);
  for (i=0;i<NUMBOBJECTS;i++)
  {
    fprintf (f,"%f %f %f %f %d %d %d %d %d %f %f %f %f %f %f %f %f\n",bobject[i].x,bobjec
t[i].y,bobject[i].z,bobject[i].r,(int)bobject[i].Red,(int)bobject[i].Green,(int)bobject[i
].Blue,(int)bobject[i].reflect,bobject[i].objecttype,bobject[i].A,bobject[i].B,bobject[i]
.C,bobject[i].top,bobject[i].bottom,bobject[i].ax,bobject[i].ay,bobject[i].az);
```

```
  }
  fprintf (f,"%d\n",NUMLIGHTS);
  for (i=100;i<(100+NUMLIGHTS);i++)
    fprintf (f,"%f %f %f %f %d\n",bobject[i].x,bobject[i].y,bobject[i].z,bobject[i].r,bo
bject[i].reflect);
  fclose(f);
  MODIFIED=FALSE;
}

clrscr()
{
  /* Is there a built-in command to do this? */

  system ("clear");
}

getmandelbrot(p,q,R,G,B)
float p,q;
int *R,*G,*B;
{
  /* Returns the color of the Mandelbrot set at point p,q */

      float boundary,x,y,oldx;
      int count;

      count=0;
      boundary=0;
      x=0;
      y=0;
      do
      {
        oldx=x;
        x=x*x-y*y+p;
        y=oldx*y+oldx*y+q;
        boundary=x*x+y*y;
        count++;
      }
      while ((boundary <= 4) && (count <= MAXCOUNT));
      if (count <= MAXCOUNT)
      {
        *R=255;
        *G=(count*20)%256;
        *B=0;
      }
      else
      {
        *R= *G= *B=0;
      }
}

changeparameters()
{
  /* Lets the user change the viewpoint, centerpoint, ambiant light and */
  /* surface */

  clrscr();
  printf ("Current eye:    <%d,%d,%d>\n",(int)EYEX,(int)EYEY,(int)EYEZ);
  printf ("Current center: <%d,%d,%d>\n",(int)CENTERX,(int)CENTERY,(int)CENTERZ);
  printf ("\nEnter new coordinates, or ENTER to keep:\n\n");
  printf ("EYEX:  ");
  gets (str);
  if (str[0]!=0) EYEX=atoi(str);
  printf ("EYEY:  ");
  gets (str);
  if (str[0]!=0) EYEY=atoi(str);
  printf ("EYEZ:  ");
  gets (str);
  if (str[0]!=0) EYEZ=atoi(str);
  printf ("CENTERX:  ");
  gets (str);
  if (str[0]!=0) CENTERX=atoi(str);
  printf ("CENTERY:  ");
  gets (str);
  if (str[0]!=0) CENTERY=atoi(str);
  printf ("CENTERZ:  ");
  gets (str);
  if (str[0]!=0) CENTERZ=atoi(str);
  printf ("\nSurface currently %d:   0=Checkerboard, 1=Mandelbrot, 2=Water\n",SURFACE);
  printf ("SURFACE:  ");
  gets (str);
  if (str[0]!=0) SURFACE=atoi(str);
  printf ("\nAmbiant light currently %d out of 255.\n",AMBIANT);
  printf ("AMBIANT:  ");
  gets (str);
  if (str[0]!=0) AMBIANT=atoi(str);
  MODIFIED=TRUE;
}
```