```
/* main.c
 * Part of a 3D Interactive Geometry Program
 * Copyright 2007 Jacob Welsh
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "SDL.h"

#include "vector.h"
#include "display.h"
#include "geom.h"

//Canvas *canv;
Space space;

int EventLoop (void) {
        SDL_Event ev;
        while (SDL_PollEvent(&ev)) {
                switch (ev.type) {
                        case SDL_KEYDOWN:
                                if (ev.key.keysym.sym == SDLK_q)
                                        return 0;
                                break;
                        case SDL_QUIT:
                                return 0;
                                break;
                        case SDL_VIDEOEXPOSE:
                                redrawAll(space.root);
                                break;
                        case SDL_VIDEORESIZE:
                                if (!sizeDisp(ev.resize.w, ev.resize.h)) {
                                        fprintf (stderr, "Failed to resize display: %s\n",
 SDL_GetError());
                                        return -1;
                                }
                                //stretch the canvas to fit the screen
                                //sizeCanvas(canv, canv->x1, canv->x2, canv->y1,
 canv->y2);
                                //redrawAll(canv);
                                redrawAll(space.root);
                                break;
                }
        }
        return 1;
}

int main (int argc, char *argv[]) {
        initDisp();
        //dispLib = DISP_SDL;
        dispLib = DISP_OPENGL;
        sizeDisp(640, 480);
        /*canv = newCanvas(-5, 5, -5, 5);
        if (!canv) {
                fprintf (stderr, "Error creating canvas!\n");
                exit(1);
        }
        Assignvfff (canv->bkgndCol, 0, 0, 0);*/
        // Create the root of the geometry tree
        Assignvfff (space.bkgndCol, 0, 0, 0);
        Assignvfff (space.eyePos, 0, 0, 1);
        Assignvfff (space.lookAt, 0, 0, 0);

        //root = newObj (GEOM_T_SPACE, NULL);
        //Assignvfff (root->obj.space.eyePos, 0, 0, 1);
        //Assignvfff (root->obj.space.lookAt, 0, 0, 0);
        //root->obj.space.zoom = 1;
        //Assignvfff (root->color, 1, 1, 1);
```



```
        space.root = newObj (GEOM_T_POINT, NULL);
        redrawAll(space.root);

        while (EventLoop()) {
                SDL_Delay (50);
        }
        return 0;
}
```

```
/* geom.h
 * Part of a 3D Interactive Geometry Program
 * Copyright 2007 Jacob Welsh
 */

#ifndef GEOM_H
#define GEOM_H

#include "vector.h"

#define GEOM_T_POINT    0
#define GEOM_T_LINE     1
#define GEOM_T_PLANE    2
#define GEOM_NUM_TYPES  3

typedef struct Point {
        vect pos;
        double t;
} Point;

typedef struct Line {
        Point *a;
        Point *b;
} Line;

typedef struct Plane {
        Point *a, *b, *c;
} Plane;

typedef struct geomObj {
        int type;
        vect color;
        union {
                Point point;
                Line line;
                Plane plane;
        } obj;
//      struct geomObj *parent;
        struct geomObj *child;
        struct geomObj *cnext; //next in the tree child LL
        struct geomObj *next; //next in the global LL
} geomObj;

geomObj *newObj (int type, geomObj *parent);

typedef struct Space {
        //Properties
        vect bkgndCol;

        //World to Screen mapping
        vect eyePos;
        vect lookAt;
        double zoom;

        //Root of geometry tree for this space
        geomObj *root;
} Space;

typedef struct Canvas {
        //Properties
        vect bkgndCol;

        //Coord to screen mapping
        Float x1, x2, y1, y2;
        Float pxPerUnitX, pxPerUnitY;

        //Root of geometry tree of this canvas
        geomObj *root;
} Canvas;
```



```
int sizeCanvas (Canvas *canv, Float x1, Float x2, Float y1, Float y2);
Canvas *newCanvas (Float x1, Float x2, Float y1, Float y2);

#endif
```

```
/* geom.c
 * Part of a 3D Interactive Geometry Program
 * Copyright 2007 Jacob Welsh
 */

#include "vector.h"
#include "geom.h"
#include "display.h"

geomObj *newObj (int type, geomObj *parent) {
        geomObj *new = malloc (sizeof(geomObj));
        if (!new) {
                fprintf (stderr, "Out of memory!");
                exit(-1);
        }
        new->type = type;
        // Join it into the tree
//      new->parent = parent;
        new->next = new->cnext = new->child = 0;
        if (parent) {
                if (parent->child) {
                        geomObj *tmp;
                        for (tmp = parent->child; tmp->next; tmp = tmp->next);
                        tmp->next = new;
                }
                else
                        parent->child = new;
        }
        return new;
}

int sizeCanvas (Canvas *canv, Float x1, Float x2, Float y1, Float y2) {
        //prevent divide by zero (enforce finite size)
        if (x1 == x2 || y1 == y2)
                return 0;
        canv->x1 = x1;
        canv->x2 = x2;
        canv->y1 = y1;
        canv->y2 = y2;
        canv->pxPerUnitX = (Float)dispW / (x2 - x1);
        canv->pxPerUnitY = (Float)dispH / (y2 - y1);
        return 1;
}

Canvas *newCanvas (Float x1, Float x2, Float y1, Float y2) {
        Canvas *canv = malloc(sizeof(Canvas));
        if (!canv) return NULL;
        if (!sizeCanvas(canv, x1, x2, y1, y2)) return NULL;
        Assignvfff(canv->bkgndCol, 0, 0, 0);
        canv->root = NULL;
        return canv;
}
```



```
/* display.h
 * Part of a 3D Interactive Geometry Program
 * Copyright 2007 Jacob Welsh
 */

#ifndef DISPLAY_H
#define DISPLAY_H

#include "SDL.h"
#include <GL/gl.h>
#include "geom.h"
#include "vector.h"

extern SDL_Surface *dispSurf;

extern int dispW, dispH;

#define DISP_OPENGL 0
#define DISP_SDL 1
extern int dispLib;

// Library-specific display functions
int initDisp(void);

int sizeDispGL(int w, int h);
int sizeDispSDL(int w, int h);

//int redrawAllGL(Canvas *canv);
//int redrawAllSDL(Canvas *canv);
int redrawAllGL(geomObj *obj);
int redrawAllSDL(geomObj *obj);

// Wrapper display functions
static inline int sizeDisp (int w, int h) {
        if (dispLib == DISP_OPENGL)
                return sizeDispGL(w, h);
        else
                return sizeDispSDL(w, h);
}
static inline int redrawAll (geomObj *obj) {
        if (dispLib == DISP_OPENGL)
                return redrawAllGL(obj);
        else
                return redrawAllSDL(obj);
}

#endif
```

```
/* display.c
 * Part of a 3D Interactive Geometry Program
 * Copyright 2007 Jacob Welsh
 */

#include "display.h"

SDL_Surface *dispSurf;

int dispW = 0, dispH = 0;

int dispLib = DISP_OPENGL;

static inline int vecToCol (vect v) {
        Uint8 r, g, b;
        if (v[0] < 0) r = 0;
        else if (v[0] > 1) r = 255;
        else r = v[0] * 255.;

        if (v[1] < 0) g = 0;
        else if (v[1] > 1) g = 255;
        else g = v[1] * 255.;

        if (v[2] < 0) b = 0;
        else if (v[2] > 1) b = 255;
        else b = v[2] * 255.;

        return SDL_MapRGB(dispSurf->format, r, g, b);
}

int coordToPx (Canvas *canv, vect v, int *x, int *y) {
        *x = (v[X] - canv->x1) * canv->pxPerUnitX;
        *y = (canv->y2 - v[Y]) * canv->pxPerUnitY;
        return 1;
}

//Drawing functions
//Return 1 = success; 0 = clipped; -1 = error
int (*drawFuncsGL[GEOM_NUM_TYPES]) (geomObj *obj);
int (*drawFuncsSDL[GEOM_NUM_TYPES]) (geomObj *obj);

// A wrapper around the function pointers, with sanity check
int drawObj (geomObj *obj) {
        if (obj && obj->type >= 0 && obj->type < GEOM_NUM_TYPES) {
                if (dispLib == DISP_OPENGL)
                        return drawFuncsGL[obj->type] (obj);
                else
                        return drawFuncsSDL[obj->type] (obj);
        }
        else
                return -1;
}

int drawPointSDL (geomObj *pt) {
//      int x, y;
//      coordToPx (canv, pt->obj.point.pos, &x, &y);
//      if (x < 0 || y < 0 || x > dispW || y > dispH) return 0;
//      ((Uint32 *)dispSurf->pixels)[x + dispW*y] = vecToCol(pt->color);
        return 0;
}
int drawPointGL (geomObj *pt) {
        glColor3dv(pt->color);
        glBegin (GL_POINTS);
                glVertex3dv (pt->obj.point.pos);
        glEnd();
        printf ("foo\n");
        return 1;
}
```



```
int drawLineSDL (geomObj *ln) {
        return 0;
}
int drawLineGL (geomObj *ln) {
        glColor3dv(ln->color);
        glBegin (GL_LINES);
                glVertex3dv (ln->obj.line.a->pos);
                glVertex3dv (ln->obj.line.b->pos);
        glEnd();
        return 1;
}

int drawPlaneSDL (geomObj *pln) {
        return 0;
}
int drawPlaneGL (geomObj *pln) {
        return 0;
}

int drawSpaceSDL (geomObj *sp) {
        return 0;
}
int drawSpaceGL (geomObj *sp) {
        glClearColor (sp->color[0], sp->color[1], sp->color[2], 1);
        glClear (GL_COLOR_BUFFER_BIT);
        return 1;
}

// Miscellaneous display functions

int initDisp(void) {
        // Initialize drawing function pointer arrays
        drawFuncsSDL[GEOM_T_POINT] = drawPointSDL;
        drawFuncsGL[GEOM_T_POINT] = drawPointGL;

        drawFuncsSDL[GEOM_T_LINE] = drawLineSDL;
        drawFuncsGL[GEOM_T_LINE] = drawLineGL;

        drawFuncsSDL[GEOM_T_PLANE] = drawPlaneSDL;
        drawFuncsGL[GEOM_T_PLANE] = drawPlaneGL;

        drawFuncsSDL[GEOM_T_SPACE] = drawSpaceSDL;
        drawFuncsGL[GEOM_T_SPACE] = drawSpaceGL;

        // Initialize SDL

        if (SDL_Init(SDL_INIT_VIDEO)) {
                fprintf (stderr, "Failed to init SDL: %s\n", SDL_GetError());
                return 0;
        }
        atexit (SDL_Quit);
        return 1;
}

int sizeDispSDL(int w, int h) {
        dispSurf = SDL_SetVideoMode(w, h, 32, SDL_SWSURFACE | SDL_RESIZABLE);
        if (!dispSurf) {
                fprintf (stderr, "Failed to get display surface: %s\n", SDL_GetError());
                return 0;
        }
        dispW = dispSurf->w;
        dispH = dispSurf->h;
        return 1;
}
int sizeDispGL(int w, int h) {
        SDL_GL_SetAttribute (SDL_GL_RED_SIZE, 8);
        SDL_GL_SetAttribute (SDL_GL_GREEN_SIZE, 8);
        SDL_GL_SetAttribute (SDL_GL_BLUE_SIZE, 8);
        SDL_GL_SetAttribute (SDL_GL_DEPTH_SIZE, 16);
```

```
        SDL_GL_SetAttribute (SDL_GL_DOUBLEBUFFER, 1);
        dispSurf = SDL_SetVideoMode(w, h, 32, SDL_OPENGL | SDL_RESIZABLE);
        if (!dispSurf) {
                fprintf (stderr, "Failed to get OpenGL surface: %s\n", SDL_GetError());
                return 0;
        }
        dispW = dispSurf->w;
        dispH = dispSurf->h;
        return 1;
}

int drawTree (geomObj *obj) {
        // Draw the desired node of the tree
        int ret = drawObj (obj);
        // Recursively traverse through each child node
        for (obj = obj->child; obj; obj = obj->next) {
                ret = ret && drawTree (obj);
        }
        // return success if all children succeeded
        return ret;
}

/*int redrawAllSDL(Canvas *canv) {
        SDL_FillRect (dispSurf, NULL, vecToCol(canv->bkgndCol));

        if (SDL_MUSTLOCK(dispSurf)) SDL_LockSurface(dispSurf);
        drawObj(canv, canv->root);
        if (SDL_MUSTLOCK(dispSurf)) SDL_UnlockSurface(dispSurf);
        SDL_UpdateRect (dispSurf, 0, 0, 0, 0);

        return 0;
}*/

int redrawAll(geomObj *obj) {
        drawTree (obj);
        if (dispLib == DISP_OPENGL) {
                SDL_GL_SwapBuffers();
        }
        return 0;
}
```



```
/*************************************************************************
  Copyright (C) 2006 Jacob Welsh <jfwelsh@verizon.net>

  This program is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 2 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the file
  COPYING for details.
 *************************************************************************
  vector.h: Defines inline functions and macros for the convenient
  manipulation of vectors.
*/

#ifndef VECTOR_H
#define VECTOR_H

#include <math.h>

typedef double Float;
typedef Float vect[3];
#define X 0
#define Y 1
#define Z 2

#define Equalvv(a,b) (Equalff(a[X], b[X]) && \
        Equalff(a[Y], b[Y]) && \
        Equalff(a[Z], b[Z]) )
#define LessThanvv(a,b) (LessThanff(a[X], b[X]) && \
        LessThanff(a[Y], b[Y]) && \
        LessThanff(a[Z], b[Z]) )
#define LessOrEqualvv(a,b) (LessOrEqualff(a[X], b[X]) && \
        LessOrEqualff(a[Y], b[Y]) && \
        LessOrEqualff(a[Z], b[Z]) )
#define GrtThanvv(a,b) (GrtThanff(a[X], b[X]) && \
        GrtThanff(a[Y], b[Y]) && \
        GrtThanff(a[Z], b[Z]) )
#define GrtOrEqualvv(a,b) (GrtOrEqualff(a[X], b[X]) && \
        GrtOrEqualff(a[Y], b[Y]) && \
        GrtOrEqualff(a[Z], b[Z]) )

#define fDotvv(a,b) (a[X]*b[X] + a[Y]*b[Y] + a[Z]*b[Z])

#define fLengthv(vec) sqrt(vec[X]*vec[X] + vec[Y]*vec[Y] + vec[Z]*vec[Z])
#define fLengthfff(a,b,c) sqrt(a*a + b*b + c*c)

inline static void Assignvv (vect dest, vect src)
{
        dest[X] = src[X];
        dest[Y] = src[Y];
        dest[Z] = src[Z];
}
inline static void Assignvf (vect dest, Float src)
{
        dest[X] = src;
        dest[Y] = src;
        dest[Z] = src;
}
inline static void Assignvfff (vect dest, Float a, Float b, Float c)
{
        dest[X] = a;
        dest[Y] = b;
        dest[Z] = c;
}
```

```
inline static void PlusEqvv (vect dest, vect src)
{
        dest[X] += src[X];
        dest[Y] += src[Y];
        dest[Z] += src[Z];
}
inline static void PlusEqvf (vect dest, Float src)
{
        dest[X] += src;
        dest[Y] += src;
        dest[Z] += src;
}

inline static void Addvvv (vect dest, vect a, vect b)
{
        dest[X] = a[X] + b[X];
        dest[Y] = a[Y] + b[Y];
        dest[Z] = a[Z] + b[Z];
}
inline static void Addvvf (vect dest, vect a, Float b)
{
        dest[X] = a[X] + b;
        dest[Y] = a[Y] + b;
        dest[Z] = a[Z] + b;
}

inline static void MinEqvv (vect subfrom, vect sub)
{
        subfrom[X] -= sub[X];
        subfrom[Y] -= sub[Y];
        subfrom[Z] -= sub[Z];
}
inline static void MinEqvf (vect subfrom, Float sub)
{
        subfrom[X] -= sub;
        subfrom[Y] -= sub;
        subfrom[Z] -= sub;
}

inline static void Subvvv (vect dest, vect a, vect b)
{
        dest[X] = a[X] - b[X];
        dest[Y] = a[Y] - b[Y];
        dest[Z] = a[Z] - b[Z];
}
inline static void Subvvf (vect dest, vect a, Float b)
{
        dest[X] = a[X] - b;
        dest[Y] = a[Y] - b;
        dest[Z] = a[Z] - b;
}

inline static void MultEqvv (vect dest, vect src)
{
        dest[X] *= src[X];
        dest[Y] *= src[Y];
        dest[Z] *= src[Z];
}
inline static void MultEqvf (vect dest, Float src)
{
        dest[X] *= src;
        dest[Y] *= src;
        dest[Z] *= src;
}

inline static void Multvvv (vect dest, vect a, vect b)
{
        dest[X] = a[X] * b[X];
        dest[Y] = a[Y] * b[Y];
```



```
        dest[Z] = a[Z] * b[Z];
}
inline static void Multvvf (vect dest, vect a, Float b)
{
        dest[X] = a[X] * b;
        dest[Y] = a[Y] * b;
        dest[Z] = a[Z] * b;
}

inline static void DivEqvv (vect dividend, vect divisor)
{
        dividend[X] /= divisor[X];
        dividend[Y] /= divisor[Y];
        dividend[Z] /= divisor[Z];
}

inline static void Divvvv (vect dest, vect a, vect b)
{
        dest[X] = a[X] / b[X];
        dest[Y] = a[Y] / b[Y];
        dest[Z] = a[Z] / b[Z];
}
inline static void Divvvf (vect dest, vect a, Float b)
{
        dest[X] = a[X] / b;
        dest[Y] = a[Y] / b;
        dest[Z] = a[Z] / b;
}

#endif //not VECTOR_H
```