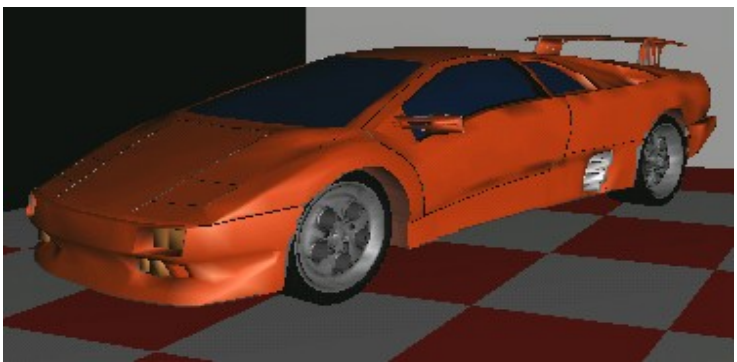
CSE 4471 – Assignment 1

VR AUTO SHOW

Matthew Conte: cs243082
Octobober 6, 2008



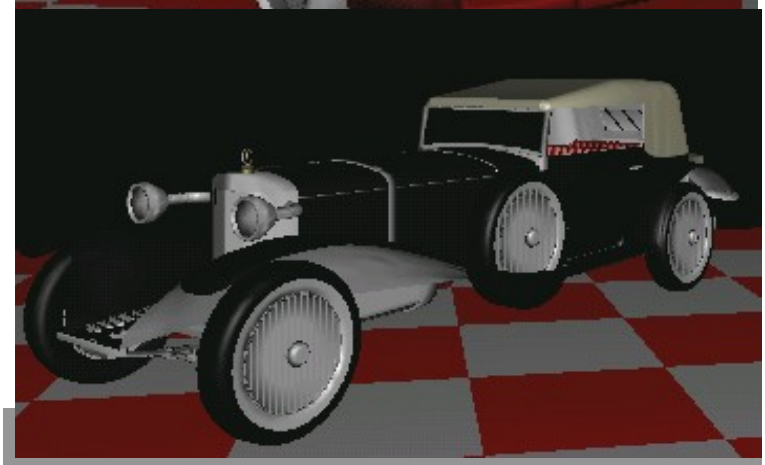
Cars of the VR Auto Show

Lamborghini Diablo – 2004

Price: $150,000 USD

Chevrolette Corvette – 2000

Price: $30,000 USD



Mercedes 28/95 PS – 1914

Price: $150, 000.00 USD

Hummer H2 - 2005

Price: $40,000.00 USD

Description:

This program simulates an interactive virtual reality auto show, where various cars are displayed and can have their rotation animation (if desired) manipulated by the user as he walks around in the scene in a first person view. The user is able to move within the environment and interact with vehicles via a joystick or keyboard.

This application can be ideal for websites for auto shows or car dealerships where potential buyers or enthusiasts can view cars in a way they couldn't as easily in reality.

Software:

This application uses the VE virtual reality library developed at York University. Also used was

a virtual environment motion (vem) library to enable character and view motion in first person originally developed by Professor Michael Jenkins which was modified specifically for this application to allow more fluent character movement with a joystick.

Hardware:

The user is able to move and interact in the environment using either a conventional keyboard or a joystick. The program is designed more effectively to use the Logitech attack 3 joystick (Fig 1) due to its conveniently placed 11 buttons.

Fig 1a: side view with trigger (button 1), on base, left to right, Button 6,7,8,9

Fig 1b: top handle buttons (left to right, top to bottom) Button 4,3,5,2

Fig 1: Logitech Attack 3 Joystick

Character/View Movement:

Action:	**Keyboard**		**Joystick**
-Forward/Backward movement (Fig 2b&c)	- **up/down** arrow keys	- tilt	forward/backward
-Pan view left/right (Fig 2d&e)	-l**eft/right** arrows keys		- tilt left/right
-Tilt view up/down (Fig 2f&g)	-n/a		- tilt forward/backward while holding trigger(button 1)
-Reset to starting position	-**r** key		- Button 7

Note: If user tilts view from original position, any future character movements or release of the joystick trigger will reset view to before tilting like in many first-person shooter games.

Figure 2: Screenshots of character/view movements

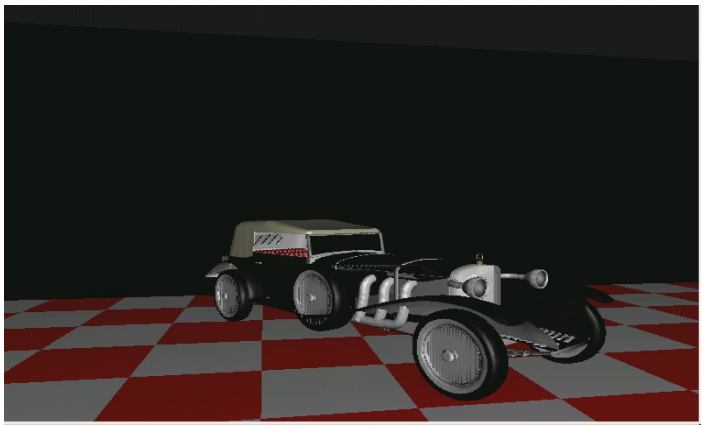


Figure 2a: original position

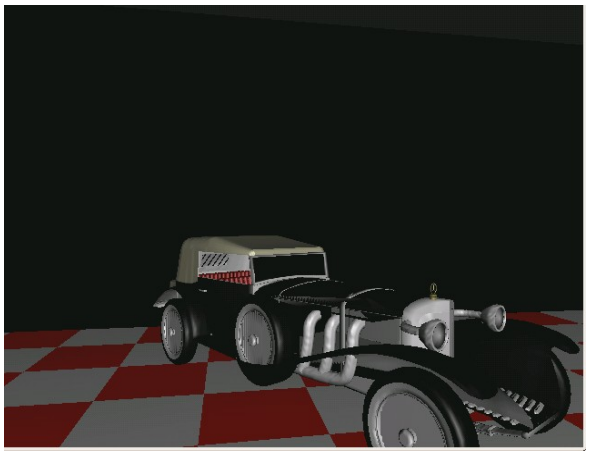


Figure 2b: Move forward

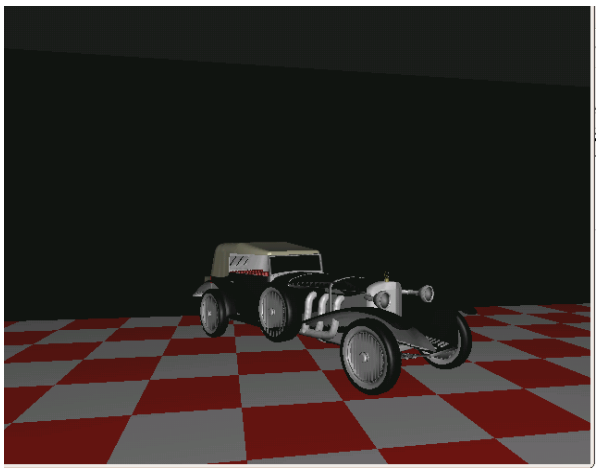


Figure 2c: Move backward

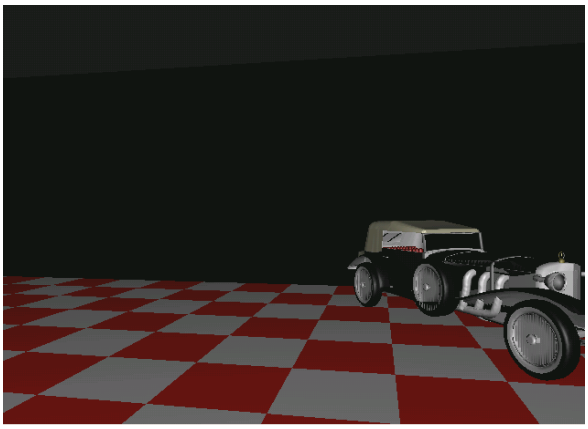


Figure 2d: Pan left

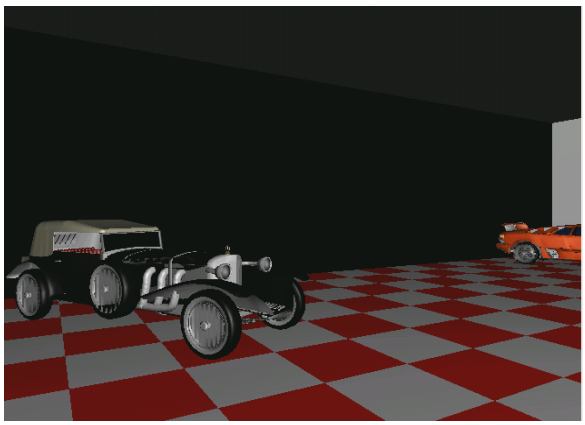


Figure 2e: Pan right

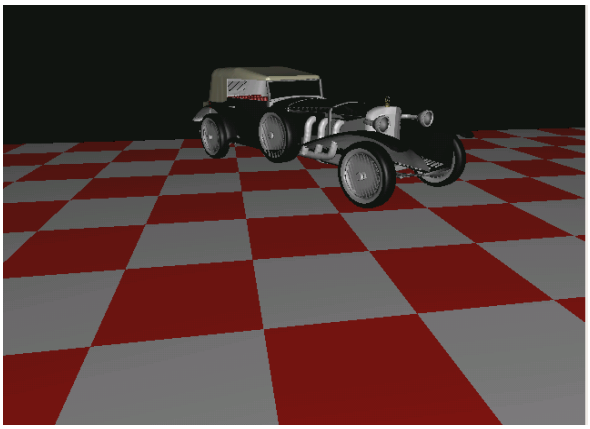


Figure 2f: Tilt down

Figure 2g: Tilt up

User/Car Interaction:

Once you're close enough to any car (~ 4 units Euclidean distance), the user is able to manipulate the rotation of the car on every axis.

When you're close enough to a vehicle, a display of the cars axis' of rotation (x,y,z) will be displayed (Figure 3). By default, all cars will initially rotate around the y-axis at a rate of 15 degrees per 1000/60 ms.

The axis that's green indicates the current manipulative axis of rotation (Figure 3). The user can then adjust the direction of rotation or disable roation altogether for the current manupulative axis of rotation and the user can also switch to another axis of rotation.

Action:	**Joystick**	**Keyboard**
– Rotate car counter-clockwise	- **Button 4**	- **a** key
– Rotate car clockwise	- **Button 5**	- **d** key
– Disable rotation of car	- **Button 3**	- **s** key
– Change current manipulative axis of rotation (order = x,y,z, x,y,z,etc.)	- **Button 2**	- **w** key

Figure 3: Car rotating amongst x,y,z axis

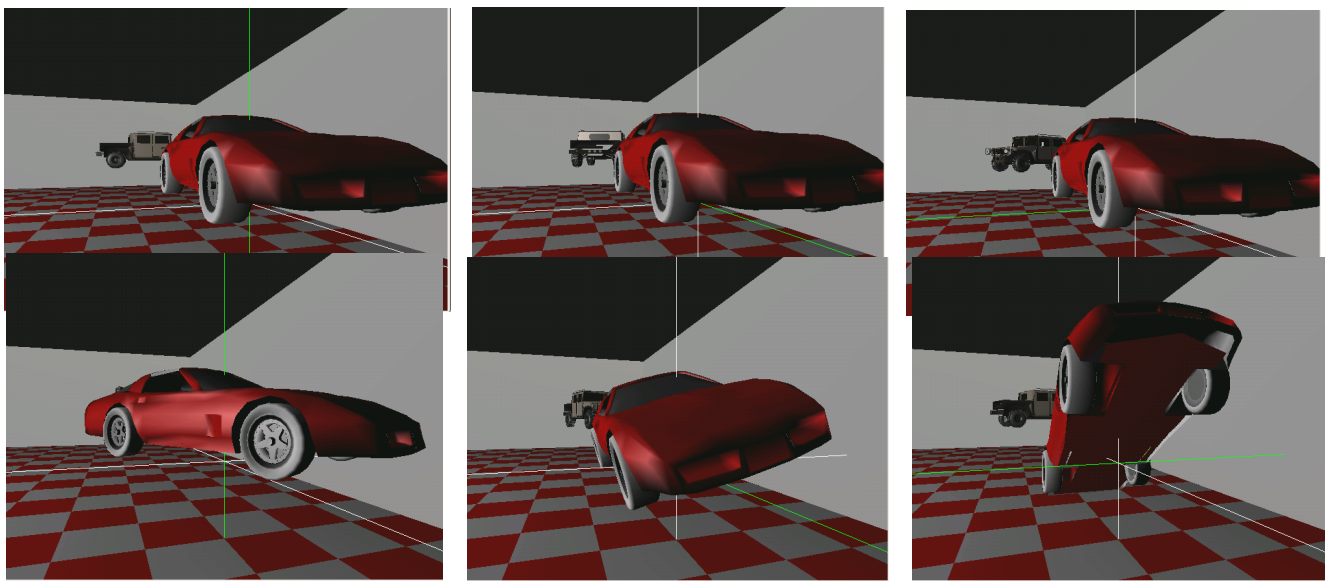


Fig 3a: Rotate about y-axis

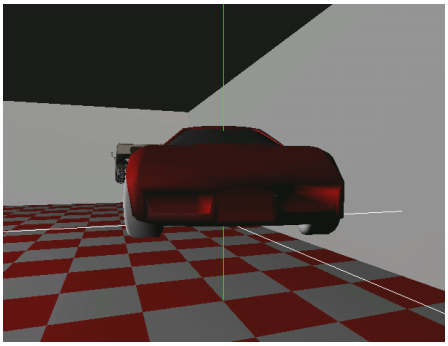


Fig 3b: Rotate about z-axis

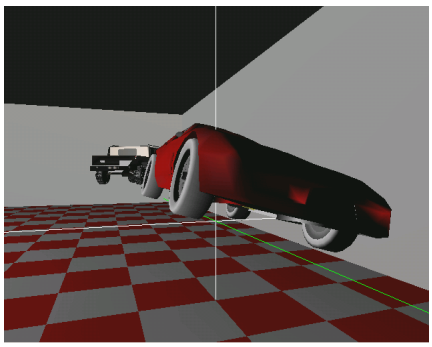
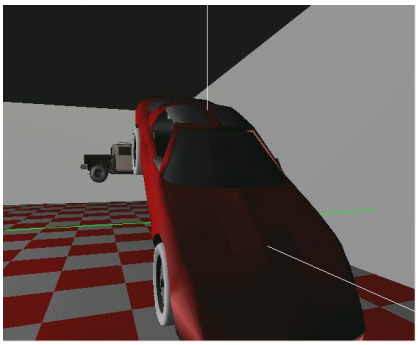


Fig 3c: Rotate about x-axis



Exiting the Program

The user can exit with a keyboard with the **Esc** key, or **Button 6** with the joystick. The user can also exit the program by walking into the door in the scene located behind you at the beginning of the application (Figure 4). The user must within 1 unit (metre) away from the door to exit through it.

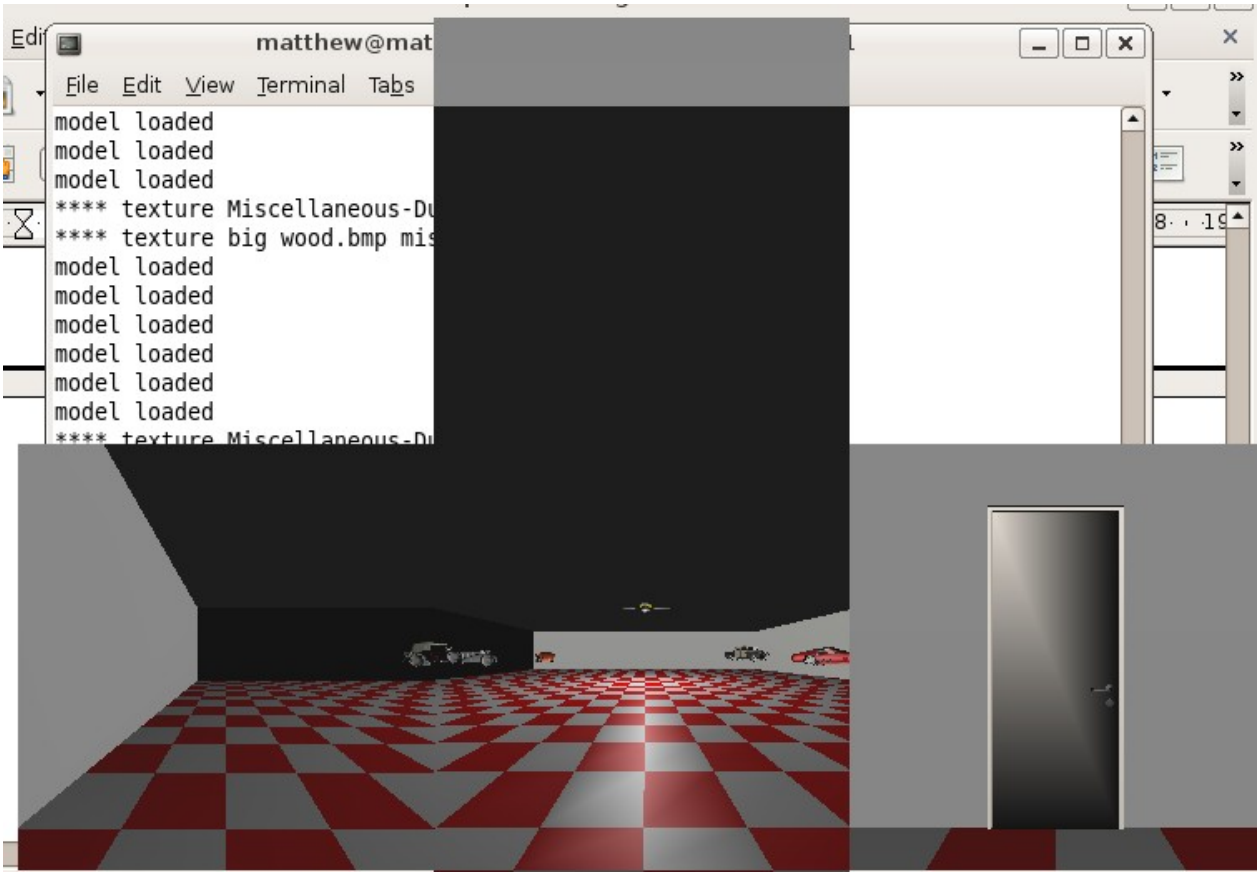


Figure 4: First person view of environment (Top = Above, Left = Left, Centre = Front, Right = Behind)

```
/************** CSE 4471 - A1 - VR Auto Show**********************
* An interactive auto show where a user moves around in a scene with several cars * and interact with them by adjust their axis of rotation
*
* October 6, 2008
*
* Matthew Conte
* cs243082
*/

# include <stdio.h>
# include <math.h>
# include <ve.h>
# include <GL/gl.h>
# include <GL/glut.h>
# include <GL/glu.h>
# define EYE_HEIGHT 1.6

# include <3ds.h>
# include <3dsRenderer.h>
# include <vem.h>

/* amount of rotation per frame */
# define ROT_SPEED 15.0

// Room dimensions (metres)
# define ROOM_WIDTH 15
# define ROOM_LENGTH 15
# define ROOM_HEIGHT 5

/* the number of milliseconds between frames (100ms)*/
#define FRAME_INTERVAL (100000/1000)

# define LIGHT

# define NUM_CARS 4

struct transform {
        float rx, ry, rz, s, tx, ty, tz;
};


static float current[4] = { 0.0, EYE_HEIGHT, 5.0, 1.0 }; //current position
static GLfloat lightPos[4] = {0.0, ROOM_HEIGHT, 0.0, 1.0}; // light source position
static float doorPos[3] = {0.0, 0.0, ROOM_LENGTH}; //position of exit doory
```

```
static int hit_car_index = -1; //index of struct array of cars that's currently "hit"


struct {
        char *dir; // path of 3ds model
        char *model; // name of file of 3ds model
        char current_axis; //current axis selected for rotation
        int rotate_x_axis; // if 1 or -1, rotate car about its x-axis
        int rotate_y_axis; // if 1 or -1, rotate car about its y-axis
        int rotate_z_axis; // if 1 or -1, rotate car about its z-axis
        struct transform t; // transformation on model
} cars[NUM_CARS] = {

        {"cars", "diablo.3ds", 'y', 0, 1, 0, {0,0,0,4.5,-12.0,1.0, -12.0}},
        {"cars", "Hummer N260907.3DS", 'y', 0, 1, 0, {0,0,0,4.5,12.0,2.0,-12.0}},
        {"cars", "Mersedes 1928 N300708.3DS", 'y', 0, 1, 0, {0,0,0,4.5,-12.0,1.0, 0.0}},
        {"cars", "Corvette.3ds", 'y', 0, 1, 0, {0,0,0,4.5,12.0,1.0,0.0}}
};


static struct t3DModel car[NUM_CARS];
static struct t3DModel light;
static struct t3DModel door;


/*
 * Setup the window on each processor
 */
static void setupwin(VeWindow *w)
{
  int i;

  static GLfloat lightamb[4] = {0.2, 0.2, 0.2, 1.0};
  static GLfloat lightdif[4] = {0.8, 0.8, 0.8, 1.0};
  static GLfloat lightspec[4] = {0.4, 0.4, 0.4, 1.0};
  static GLfloat loc[4] = {0.0, 10.0, 0.0, 1.0};
  static GLfloat diffuseMaterial[4] = {0.8, 0.8, 0.8, 1.0};
  static GLfloat mat_specular[4] = {1.0, 1.0, 1.0, 1.0};

  glEnable(GL_DEPTH_TEST);
  glEnable(GL_BLEND);
  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);


# ifdef LIGHT
  glEnable(GL_LIGHTING);
```

```
  glEnable(GL_NORMALIZE);
  glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, 1);
  glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, 1);
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lightamb);

  /* one other light source */
  glLightfv(GL_LIGHT0, GL_AMBIENT, lightamb);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, lightdif);
  glLightfv(GL_LIGHT0, GL_SPECULAR, lightspec);
  glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
  glEnable(GL_LIGHT0);

  glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuseMaterial);
  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialf(GL_FRONT, GL_SHININESS, 25.0);
  glColorMaterial(GL_FRONT, GL_DIFFUSE);
  glEnable(GL_COLOR_MATERIAL);
#endif

  glClearColor(0,0,0,0);
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

  glEnable(GL_TEXTURE_2D);

        // Import car models
        for (i=0; i<NUM_CARS;i++)
        {
          if(Import3DS(&car[i], cars[i].dir, cars[i].model)< 0)
            fprintf(stderr,"model load fails\n");
          else
            fprintf(stderr, "model loaded\n");
        }


        // Import static models (decorations)
        if (Import3DS(&light, "static", "fan object.3ds") < 0)
                fprintf(stderr, "model load fails\n");
        else
                fprintf(stderr, "model loaded\n");


        if (Import3DS(&door, "static", "add073.3DS") < 0)
                fprintf(stderr, "model load fails\n");
        else
                fprintf(stderr, "model loaded\n");
}
```

```
/*
 * On each processor, generate the appropriate redisplay
 */
static void display(VeWindow *w, long tm, VeWallView *wv)
{
  int x, z, c;
# ifdef LIGHT
  GLfloat loc[4] = {0.0, 4.0, 0.0, 1.0};

  glLightfv(GL_LIGHT0, GL_POSITION, loc);
# endif

  glClear(GL_DEPTH_BUFFER_BIT|GL_COLOR_BUFFER_BIT);
  glDisable(GL_TEXTURE_2D);

  //draw checkerboard floor here
  c = 1;
  for(x= -ROOM_WIDTH; x<ROOM_WIDTH; x++){
    for(z= -ROOM_LENGTH; z<ROOM_LENGTH;z++) {
      if(c)
          glColor3ub(255,0,0);
      else
          glColor3ub(255,255,255);
      c = !c;
      glBegin(GL_QUADS);
      glNormal3f(0,1,0);
      glVertex3f(x, 0, z);
      glVertex3f(x, 0, z+1);
      glVertex3f(x+1, 0, z+1);
      glVertex3f(x+1, 0, z);
      glEnd ();
    }
    c = !c;
  }

          // five grey walls (texture with "VR Auto Show")

          // left wall
          glColor3ub(225, 225, 221);
          glBegin(GL_QUADS);
          glNormal3f(1,0,0);
          glVertex3f(-ROOM_WIDTH, 0, -ROOM_LENGTH);
          glVertex3f(-ROOM_WIDTH, 0, ROOM_LENGTH);
          glVertex3f(-ROOM_WIDTH, 5, ROOM_LENGTH);
          glVertex3f(-ROOM_WIDTH, 5, -ROOM_LENGTH);
```

```
glEnd();

//right wall
glColor3ub(225, 225, 221);
glBegin(GL_QUADS);
glNormal3f(-1,0,0);
glVertex3f(ROOM_WIDTH, 0, -ROOM_LENGTH);
glVertex3f(ROOM_WIDTH, 0, ROOM_LENGTH);
glVertex3f(ROOM_WIDTH, 5, ROOM_LENGTH);
glVertex3f(ROOM_WIDTH, 5, -ROOM_LENGTH);
glEnd();

// back wall
glColor3ub(225, 225, 221);
glBegin(GL_QUADS);
glNormal3f(0,0,1);
glVertex3f(-ROOM_WIDTH, 0, -ROOM_LENGTH);
glVertex3f(ROOM_WIDTH, 0, -ROOM_LENGTH);
glVertex3f(ROOM_WIDTH, 5, -ROOM_LENGTH);
glVertex3f(-ROOM_WIDTH, 5, -ROOM_LENGTH);
glEnd();

// ceiling
glColor3ub(225, 225, 221);
glBegin(GL_QUADS);
glNormal3f(0,-1,0);
glVertex3f(-ROOM_WIDTH, 5, ROOM_LENGTH);
glVertex3f(-ROOM_WIDTH, 5, -ROOM_LENGTH);
glVertex3f(ROOM_WIDTH, 5, -ROOM_LENGTH);
glVertex3f(ROOM_WIDTH, 5, ROOM_LENGTH);
glEnd();

// back wall
glColor3ub(225, 225, 221);
glBegin(GL_QUADS);
glNormal3f(0,0,1);
glVertex3f(-ROOM_WIDTH, 0, ROOM_LENGTH);
glVertex3f(ROOM_WIDTH, 0, ROOM_LENGTH);
glVertex3f(ROOM_WIDTH, 5, ROOM_LENGTH);
glVertex3f(-ROOM_WIDTH, 5, ROOM_LENGTH);
glEnd();



//////////////////////////////////////////////////////////////////
// Display cars and axis of rotation of interactive car (if any)
```

```
//////////////////////////////////////////////////////////////

int i = 0;

for (i=0; i<NUM_CARS; i++)
{
 // if car is close enough to user,
 // display current rotable axis green and others white
 if (i == hit_car_index)
 {
        if (cars[i].current_axis == 'x')
                glColor3f(0, 255, 0);
        else
                glColor3f(255, 255, 255);

        glBegin(GL_LINES);
        glVertex3f(cars[i].t.tx - 4.0f,
                   cars[i].t.ty,
                   cars[i].t.tz);
        glVertex3f(cars[i].t.tx + 4.0f,
                   cars[i].t.ty,
                   cars[i].t.tz);
        glEnd();


        if (cars[i].current_axis == 'y')
                glColor3f(0, 255, 0);
        else
                glColor3f(255, 255, 255);

        glBegin(GL_LINES);
        glVertex3f(cars[i].t.tx,
                   cars[i].t.ty - 4.0f,
                   cars[i].t.tz);
        glVertex3f(cars[i].t.tx,
                   cars[i].t.ty + 4.0f,
                   cars[i].t.tz);
        glEnd();



        if (cars[i].current_axis == 'z')
                glColor3f(0, 255, 0);
        else
                glColor3f(255, 255, 255);
```

```
            glBegin(GL_LINES);
            glVertex3f(cars[i].t.tx,
                       cars[i].t.ty,
                       cars[i].t.tz - 4.0f);
            glVertex3f(cars[i].t.tx,
                       cars[i].t.ty,
                       cars[i].t.tz + 4.0f);
            glEnd();
    }


    glPushMatrix();

    glTranslatef(cars[i].t.tx, cars[i].t.ty, cars[i].t.tz );

    glScalef(cars[i].t.s, cars[i].t.s, cars[i].t.s);

    glRotatef(cars[i].t.rz, 0.0f, 0.0f, 1.0f);
  //glRotatef(cars[i].t.rz, 0.0f, 0.0f, car[i].center_z);
    //glRotatef(cars[i].t.rz, z_rotate_vector[0], z_rotate_vector[1], z_rotate_vector[2]);

    glRotatef(cars[i].t.ry, 0.0f, 1.0f, 0.0f);
    //glRotatef(cars[i].t.ry, 0.0f, car[i].center_y, 0.0f);
    //glRotatef(cars[i].t.ry, y_rotate_vector[0], y_rotate_vector[1], y_rotate_vector[2]);

    glRotatef(cars[i].t.rx, 1.0f, 0.0f, 0.0f);
    //glRotatef(cars[i].t.rx, car[i].center_x, 0.0f, 0.0f);
    //glRotatef(cars[i].t.rx, x_rotate_vector[0], x_rotate_vector[1], x_rotate_vector[2]);


    glScalef(car[i].scale, car[i].scale, car[i].scale);

    //?
    //glTranslatef(-car[i].center_x,-car[i].center_y,-car[i].center_z);
    Render3DS(&car[i]);

    glPopMatrix();
   }


  ////////////////////////////////////////////////////////////////////
  // Include ceiling light
  ////////////////////////////////////////////////////////////////////

  glPushMatrix();
```

```
	glTranslatef(lightPos[0], lightPos[1]-0.5, lightPos[2]);

	glScalef(3*light.scale, 3*light.scale, 3*light.scale);

	Render3DS(&light);

	glPopMatrix();



	glPushMatrix();

	glTranslatef(doorPos[0], doorPos[1], doorPos[2]);

	glScalef(3*door.scale, 3*door.scale, 3*door.scale);

	Render3DS(&door);

	glPopMatrix();

}

static int exitcback(VeDeviceEvent *e, void *arg) {
  exit(0);
  /*NOTREACHED*/
  return -1;
}

/*
Timer call back
*/
static void timer_callback(void *unused)
{
	int i;
	for (i=0; i < NUM_CARS; i++)
	{
		// rotate cars here (if applicable)
		if (cars[i].rotate_x_axis != 0)
			cars[i].t.rx += cars[i].rotate_x_axis*ROT_SPEED;

		if (cars[i].rotate_y_axis != 0)
			cars[i].t.ry += cars[i].rotate_y_axis*ROT_SPEED;

		if (cars[i].rotate_z_axis != 0)
			cars[i].t.rz += cars[i].rotate_z_axis*ROT_SPEED;
```

```
	}
	vePostRedisplay();
	veAddTimerProc(FRAME_INTERVAL, timer_callback, (void *)NULL);
}


static void notifier()
{
  VEM_get_pos(&current[0], &current[1], &current[2]);
}


static int collision(float *pos, float *from)
{
	int i;

	// Check for car collision (4 units away Euclidean distance)
	for (i=0; i< NUM_CARS; i++)
	{
		if ( sqrt( ((pos[0] - cars[i].t.tx) * (pos[0] - cars[i].t.tx)) + ((pos[2] - cars[i].t.tz) * (pos[2] -
cars[i].t.tz))) <= 4.0f)
		{
			printf("CAR COLLISION with %s\n", cars[i].model);
			hit_car_index = i;
			return 1;
		}
	}

	// Check for wall collision
	if ( fabs(pos[0])  >= ROOM_WIDTH || fabs(pos[2]) >= ROOM_LENGTH || fabs(pos[1]) >=
ROOM_HEIGHT)
	{
		printf("WALL COLLISION\n");
		hit_car_index = -1;
		return 1;
	}

	// Check if you run into the door, if so, exit program
	if ( sqrt( ((pos[0] - doorPos[0]) * (pos[0] - doorPos[0])) + ((pos[2] - doorPos[2]) * (pos[2] -
doorPos[2]))) <= 1.0f)
	{
		  exit(0);
	}

	printf("NO COLLISION\n");
	hit_car_index = -1;
```

```
	return 0;
}

// Switch current manipulative axis rotation here
static int axisChange(VeDeviceEvent *e, void *arg) {

	if (hit_car_index > -1)
	{
		if (cars[hit_car_index].current_axis == 'z')
			cars[hit_car_index].current_axis = 'x';
		else if (cars[hit_car_index].current_axis == 'y')
			cars[hit_car_index].current_axis = 'z';
		else if (cars[hit_car_index].current_axis == 'x')
			cars[hit_car_index].current_axis = 'y';
	}

	return 0;
}

// invoke counter-clockwise rotation for current manipulative axis of a car
static int ccw(VeDeviceEvent *e, void *arg) {

	if (hit_car_index > -1)
	{
		if (cars[hit_car_index].current_axis == 'x')
			cars[hit_car_index].rotate_x_axis = 1;
		else if (cars[hit_car_index].current_axis == 'y')
			cars[hit_car_index].rotate_y_axis = 1;
		else if (cars[hit_car_index].current_axis == 'z')
			cars[hit_car_index].rotate_z_axis = 1;
	}

	return 0;
}

// invoke clockwise rotation for current manipulative axis of a car
static int cw(VeDeviceEvent *e, void *arg) {

	if (hit_car_index > -1)
	{
		if (cars[hit_car_index].current_axis == 'x')
			cars[hit_car_index].rotate_x_axis = -1;
		else if (cars[hit_car_index].current_axis == 'y')
			cars[hit_car_index].rotate_y_axis = -1;
		else if (cars[hit_car_index].current_axis == 'z')
			cars[hit_car_index].rotate_z_axis = -1;
```

```
	}

	return 0;
}

// invoke no rotation for current manipulative axis of a car
static int stop(VeDeviceEvent *e, void *arg) {

	if (hit_car_index > -1)
	{
		if (cars[hit_car_index].current_axis == 'x')
			cars[hit_car_index].rotate_x_axis = 0;
		else if (cars[hit_car_index].current_axis == 'y')
			cars[hit_car_index].rotate_y_axis = 0;
		else if (cars[hit_car_index].current_axis == 'z')
			cars[hit_car_index].rotate_z_axis = 0;
	}

	return 0;
}


int main(int argc, char **argv)
{
  veInit(&argc, argv);

  veSetOption("depth", "1");


  veRenderSetupCback(setupwin);
  veRenderCback(display);

  //veMPAddStateVar(0, &globalState, sizeof(globalState), VE_MP_AUTO);

  /* certain things only happen on the master machine */
  if (veMPIsMaster()) {

    /* Setup processing callbacks */
    VEM_default_bindings();
    VEM_check_collisions(collision);

    VEM_initial_position(0.0f, EYE_HEIGHT, ROOM_LENGTH-2);
    veDeviceAddCallback(exitcback, NULL, "exit");

    veDeviceAddCallback(cw, NULL, "cw");
    veDeviceAddCallback(ccw, NULL, "ccw");
```

```
    veDeviceAddCallback(stop, NULL, "stop");
    veDeviceAddCallback(axisChange, NULL, "axisChange");

    notifier();
    VEM_notify(notifier);
  }

  veAddTimerProc(0,timer_callback,NULL);

  txmSetRenderer(NULL, txmOpenGLRenderer());
  txmSetMgrFlags(NULL, TXM_MF_SHARED_IDS);

  veRun();
}
```

```
                                                    ############
                                                    ##devices file##
                                                    ############
use keyboard

use joystick joystick {
  optional 1
}


filter joytsick.button1 {
        $e    rename axisChange
}

filter joystick.button0 {
        $e    rename switch_pan
}

filter joystick.button3 {
        $e    rename ccw
}

filter joystick.button2 {
        $e    rename stop
}

filter joystick.button4 {
        $e    rename cw
}

filter joystick.button8 {
        $e    rename tilt_inc
}

filter joystick.button7 {
        $e    rename tilt_dec
}

filter joystick.button6 {
        $e    rename reset
}

filter joystick.button5 {
        $e    rename exit
}
```

```
# axis 1 is the forward
filter joystick.axis1 {
        $e    rename moving
}

# axis 0 is the paning
filter joystick.axis0 {
        $e    rename paning
}



filter keyboard.s {
  $e rename stop
}

filter keyboard.Right {
  $e rename pan_dec}

filter keyboard.Left {
  $e rename pan_inc}

filter keyboard.d {
  $e rename cw}

filter keyboard.a {
  $e rename ccw}

filter keyboard.w {
  $e rename axisChange
}

filter keyboard.Up {
  $e rename forward}

filter keyboard.Down {
  $e rename backward}

filter keyboard.r {
  $e rename reset
}

filter keyboard.Escape { exit }

filter *.* {
```

```
  $e dump
}
```

```
/*
 * vem is a very simple library to allow motion within ve (ve motion).
 *
 * Basically it defines a set of static motion operations so you can move about.
 * There are two callbacks that are of interest. collision, which is called to
 * allow the user to check for collisions, and notifier, which lets the user
 * know that the user has moved and a redisplay will happen). This is useful
 * if you are moving things around with the user.
 *
 * Michael Jenkin, June 2007.
 *
 * Modified by Matthew Conte, October 2008
 */

# include <math.h>
# include <stdio.h>

# include <ve.h>

# include "vem.h"

static float loc0[4] = { 0.0, 0.0, 0.0, 1.0 };
static float loc[4] = { 0.0, 0.0, 0.0, 1.0 };
static float dir[3] = { 0.0, 0.0, -1.0 };
static float up[3] = { 0.0, 1.0, 0.0 };
static float right[3] = {1.0, 0.0, 0.0};


/* NEW variables */
static int pan_clicked = -1; // 1 if in pan mode, -1 otherwise
static float total_tilt; // total amount of tilting from original head posiion
static float total_pan; // total amount of panning from original head position


/* NEW method */
// Enables/Disables panning mode
static int switch_pan()
{
        pan_clicked *= -1;
        return 0;
}


static int defaultCollision(float *pos, float *from)
{
/*   printf("Checking collision with [%f %f %f]\n",pos[0],pos[1],pos[2]); */
```

```
  return 0;
}

static int (*collision)() = defaultCollision;

static void defaultNotifier() {}
static void (*notifier)() = defaultNotifier;

/*
 * Rotate a vector by an angle (in degrees) about a given axis
 */
static void rotarb(float *axis, float ang, float *val) {
  float nval[3];
  float m[3][3];
  float d, sn, cs;
  int i;
  ang *= M_PI / 180.0;
  sn = sin(ang);
  cs = cos(ang);

  /* This matrix from Foley, van Dam - (5.79), with a correction */
  m[0][0] = axis[0]*axis[0] + cs*(1 - axis[0]*axis[0]);
  m[0][1] = axis[0]*axis[1]*(1 - cs) - axis[2]*sn;
  m[0][2] = axis[0]*axis[2]*(1 - cs) + axis[1]*sn;
  m[1][0] = axis[0]*axis[1]*(1 - cs) + axis[2]*sn;
  m[1][1] = axis[1]*axis[1] + cs*(1 - axis[1]*axis[1]);
  m[1][2] = axis[1]*axis[2]*(1 - cs) - axis[0]*sn;
  m[2][0] = axis[0]*axis[2]*(1 - cs) - axis[1]*sn;
  m[2][1] = axis[1]*axis[2]*(1 - cs) + axis[0]*sn;
  m[2][2] = axis[2]*axis[2] + cs*(1 - axis[2]*axis[2]);
  d = 0.0;
  for(i = 0; i < 3; i++) {
    nval[i] = m[i][0]*val[0]+m[i][1]*val[1]+m[i][2]*val[2];
    d += nval[i]*nval[i];
   }
  if (d != 0.0) {
    d = sqrt(d);
    nval[0] /= d; nval[1] /= d; nval[2] /= d;
   }
  for(i = 0; i < 3; i++)
    val[i] = nval[i];

}

static void pan(float a) {
  /* rotate axes around up */
```

```
  rotarb(up,a,dir);
  rotarb(up,a,right);
}

static void tilt(float a) {
  /* rotate axes around right */
  rotarb(right,a,up);
  rotarb(right,a,dir);
}

static void twist(float a) {
  /* rotate axes around dir */
  rotarb(dir,a,up);
  rotarb(dir,a,right);
}


static void move(float d) {
  int i;
  float tloc[3];

  for(i = 0; i < 3; i++)
    tloc[i] = loc[i] + dir[i]*d;


  if(!(*collision)(tloc,loc)) {
    for(i=0;i<3;i++)
      loc[i] = tloc[i];
   }

}


/*
 * Reset the viewer to the ve origin
 */
static void pose_reset()
{
  int i;

  for(i=0;i<3;i++)
    loc[i] = loc0[i];
  loc[3] = 1.0;
  dir[0] = dir[1] = 0.0;
  dir[2] = -1.0;
  up[0] = up[2] = 0.0; up[1] = 1.0;
```

```
  right[0] = 1.0; right[1] = 0.0; right[2] = 0.0;
}

static void pose_update(void) {
  VeFrame *f;
  int i;
  f = veGetOrigin();
  for(i = 0; i < 3; i++) {
    f->loc.data[i] = loc[i];
    f->dir.data[i] = dir[i];
    f->up.data[i] = up[i];
  }
  /*  f->loc.data[1] += 1.145; */ // eye offset in IVY
  (*notifier)();
  vePostRedisplay();
}

/* NEW method*/
// Moves the user with a valuator, ideal for axis movements.
static int moving(VeDeviceEvent *e, void *arg) {
  printf("MOVING\n");
  printf("Event %s %s\n",e->device,e->elem);
  if(VE_EVENT_TYPE(e) != VE_ELEM_VALUATOR) {
    fprintf(stderr,"driver: internal logic error...that should be a valuator\n");
    return(0);
  }
  VeDeviceE_Valuator *v = VE_EVENT_VALUATOR(e);
  float val = v->value;
  float min = v->min;
  float max = v->max;
  printf("%f <= %f <= %f\n",min,val,max);

  // if not in or escaping "panning" mode, tilt to original view before moving
  if (pan_clicked == -1)
  {
            tilt(-total_tilt);
            pan(-total_pan);
            total_tilt = 0;
            total_pan = 0;

            move(-val);
  }
  // if in "panning" mode, tilt forward will tilt view instead of move
  else if (pan_clicked == 1)
  {
            total_tilt += val;
```

```
            tilt(val);
  }

  pose_update();
}

/* NEW method */
// pans the users view with a valuator, ideal for axis movements
static int paning(VeDeviceEvent *e, void *arg) {
  printf("PANING\n");
  printf("Event %s %s\n",e->device,e->elem);
  if(VE_EVENT_TYPE(e) != VE_ELEM_VALUATOR) {
    fprintf(stderr,"driver: internal logic error...that should be a valuator\n");
    return(0);
  }
  VeDeviceE_Valuator *v = VE_EVENT_VALUATOR(e);
  float val = v->value;
  float min = v->min;
  float max = v->max;
  printf("%f <= %f <= %f\n",min,val,max);

  if (pan_clicked == 1)
          total_pan += val;

  pan(-val);
  pose_update();
}

/* NEW method */
// tilt the users view with a valuator, ideal for axis movements
static int tilting(VeDeviceEvent *e, void *arg) {
  printf("TILTING\n");
  printf("Event %s %s\n",e->device,e->elem);
  if(VE_EVENT_TYPE(e) != VE_ELEM_VALUATOR) {
    fprintf(stderr,"driver: internal logic error...that should be a valuator\n");
    return(0);
  }
  VeDeviceE_Valuator *v = VE_EVENT_VALUATOR(e);
  float val = v->value;
  float min = v->min;
  float max = v->max;
  printf("%f <= %f <= %f\n",min,val,max);

  tilt(-val);
  pose_update();
}
```

```
int VEM_pan_inc(VeDeviceEvent *e, void *arg) { pan(10.0); pose_update(); return 0;}
int VEM_pan_dec(VeDeviceEvent *e, void *arg) { pan(-10.0); pose_update(); return 0;}
int VEM_tilt_inc(VeDeviceEvent *e, void *arg) { tilt(10.0); pose_update(); return 0;}
int VEM_tilt_dec(VeDeviceEvent *e, void *arg) { tilt(-10.0); pose_update(); return 0;}
int VEM_twist_inc(VeDeviceEvent *e, void *arg) { twist(10.0); pose_update(); return 0;}
int VEM_twist_dec(VeDeviceEvent *e, void *arg) { twist(-10.0); pose_update();return 0;}
int VEM_forward(VeDeviceEvent *e, void *arg) { move(0.5); pose_update(); return 0;}
int VEM_backward(VeDeviceEvent *e, void *arg) { move(-0.5); pose_update(); return 0;}
int VEM_reset(VeDeviceEvent *e, void *arg) { pose_reset(); pose_update(); return 0;}

void VEM_default_bindings()
{
  veDeviceAddCallback(VEM_pan_inc, NULL, "pan_inc");
  veDeviceAddCallback(VEM_pan_dec, NULL, "pan_dec");
  veDeviceAddCallback(VEM_tilt_inc, NULL, "tilt_inc");
  veDeviceAddCallback(VEM_tilt_dec, NULL, "tilt_dec");
  veDeviceAddCallback(VEM_twist_inc, NULL, "twist_inc");
  veDeviceAddCallback(VEM_twist_dec, NULL, "twist_dec");
  veDeviceAddCallback(VEM_forward, NULL, "forward");
  veDeviceAddCallback(VEM_backward, NULL, "backward");
  veDeviceAddCallback(VEM_reset, NULL, "reset");
  veDeviceAddCallback(moving, NULL, "moving");
  veDeviceAddCallback(tilting, NULL, "tilting");
  veDeviceAddCallback(paning, NULL, "paning");
  veDeviceAddCallback(switch_pan, NULL, "switch_pan");
}

void VEM_initial_position(float x, float y, float z)
{
  loc0[0] = loc[0] = x;
  loc0[1] = loc[1] = y;
  loc0[2] = loc[2] = z;
  pose_update();
}


void VEM_no_collisions()
{
  collision = defaultCollision;
}

void VEM_check_collisions(int (*fn)())
{
  collision = fn;
```

```
}

void VEM_notify(void (*fn)())
{
  notifier = fn;
}


void VEM_get_pos(float *x, float *y, float *z) {
  *x = loc[0];
  *y = loc[1];
  *z = loc[2];
}
```