CS3241 Computer Graphics (2023/2024 Semester 1)

Lab Assignment 2

Release Date: 15 September 2023, Friday

Submission Deadline: 1 October 2023, Sunday, 11:59 PM

LEARNING OBJECTIVES

OpenGL viewing, OpenGL transformations, hierarchical modeling, and animation. After completing the programming assignment, you should have learned

- how to set up the view transformation (camera position and orientation) in OpenGL,
- how to set up perspective viewing in OpenGL,
- how to use the OpenGL transformations for modeling, and
- how to use the OpenGL transformations for animation.

TASKS

From the **`Canvas > CS3241 > Files > Lab Assignments`** folder, download the ZIP file **`Lab2_todo_(*).zip`**.

You are provided with an **incomplete** C++ application program **`main.cpp`**, and your job is to complete it according to the requirements.

You may try the **completed executables: `main_done.exe`** (for Windows), and **`main_done`** (for macOS) found in the same ZIP file. (On macOS, you may need to use the command **`sudo chmod +x main_done`** to give the file execute permission before running it.)

Please read the instructions shown in the console/terminal window to learn how to operate the program. When you run the program, you should see a spherical planet at the center of the window. There are a dozen cars moving on the planet surface, and each car moves in a different ***great circle*** (you can search the web to find out what a ***great circle*** is), and they have different speeds and colors.

You may try the followings on the program:

- resize the window and see what happens,
- press the Up, Down, Left or Right arrow key to change the camera's position,
- press the Page Up or Page Down key to change the camera's distance from the planet,
- press the 'P' key to pause/resume the animation of the cars.

You will notice that the camera is always looking at the center of the planet. With respect to the planet, the camera's position can be expressed as latitude and longitude, and its distance from the planet's center. When the Left or Right arrow key is pressed, the camera's longitude decreases or increases, respectively; and when the Down or Up arrow key is pressed, the camera's latitude decreases or increases, respectively. Note that the camera's up-vector is always pointing north.

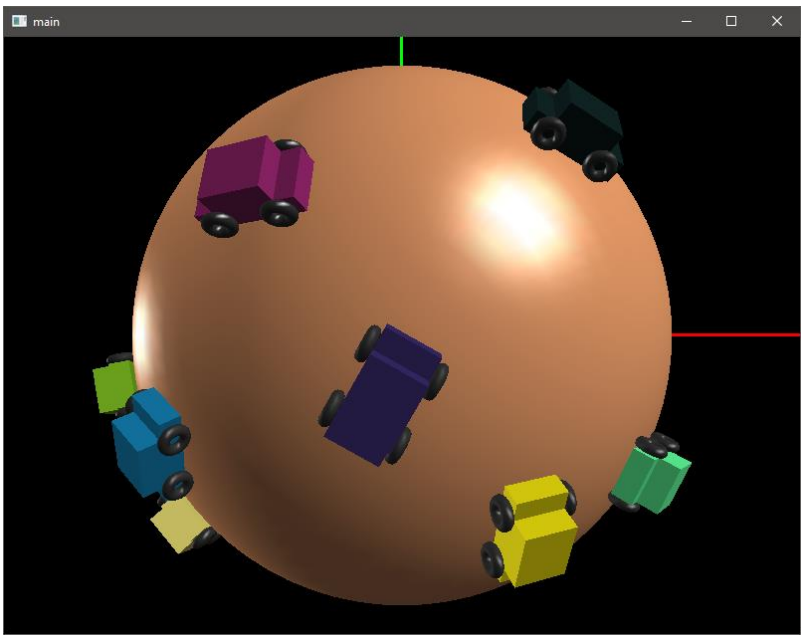

Figure 1

Follow the following instructions to complete **`main.cpp`** as required. You must not add or change any other files.

1) Study the source program very carefully.

2) Complete the **`DrawOneCar()`** function. The function must draw the car using only GLUT functions such as **`glutSolidCube()`**, **`glutSolidTorus()`**, **`glutSolidCone()`**, and **`glutSolidSphere()`**. **You should not directly use any OpenGL geometric primitive**. You should make use of the OpenGL 3D transformation functions to help you resize, orientate and position the parts. The functions **`glPushMatrix()`** and **`glPopMatrix()`** are very helpful for you to save and restore the current transformation before and after drawing each part. You can design your cars any way you like as long as they look like cars. More details about the GLUT functions can be found at https://www.opengl.org/resources/libraries/glut/spec3/spec3.html.

3) Complete the **`DrawAllCars()`** function. This function draws each car at the correct position on its great circle. Note that any great circle on a sphere centered at the origin can be defined as follows. Let C be the great circle in the $y = 0$ plane, and let v be a vector in the x-z plane, and let θ be an angle. If we rotate C about v by θ, we get another great circle of the sphere. All great circles of the sphere can be obtained by varying v and θ.

4) Set up the correct perspective viewing volume in the **`MyDisplay()`** function. You should use the **`gluPerspective()`** function. The near and far planes should be set near the planet's surface, yet still do not clip off any part of the planet and cars. The near and far planes should vary with the eye's distance from the planet's center. You should make use of the value of the predefined constant **`CLIP_PLANE_DIST`** to position your near and far planes.

5) Set up the correct view transformation (camera's position and orientation) in the **`MyDisplay()`** function. You may use the **`gluLookAt()`** function, or you can use an alternative method.

6) Complete the **`MyTimer()`** function. You should use the GLUT timer callback to control the speed of the animation by maintaining a constant frame rate (**`DESIRED_FPS`**). Refer to https://www.opengl.org/resources/libraries/glut/spec3/node64.html to find out more about the GLUT function **`glutTimerFunc()`**.

More detailed instructions can be found in the unfinished program code. You may add additional constants, global variables and functions to the given program.

DO NOT HARD-CODE VALUES. You should write your code in such a way that when the values of the named constants (defined in the beginning of the program) are changed to other valid values, your program should function accordingly. For example, if the car's size is changed, the tyre size should vary proportionally.

A Visual Studio 2017 solution **`main.sln`** (or Xcode project **`main.xcodeproj`** on macOS) is provided for you to build the executable program. In this assignment, **you are not required and must not change any other C/C++ source files** besides **`main.cpp`**.

Besides GLUT (or FreeGLUT), you should not use any other third-party libraries. Your code must compile with either the MSVC++ 2017 (or newer) compiler on Windows, or Clang on macOS.

GRADING

The maximum marks for this programming assignment is **100**, and it constitutes **7%** of your total marks for the module. The marks are allocated as follows:

- **30 marks** — drawing of a car in the **`DrawOneCar()`** function.
- **30 marks** — putting each car at its correct position on its great circle in the **`DrawAllCars()`** function.
- **10 marks** — setting up the correct **perspective viewing volume** with tight and varying near and far planes.
- **25 marks** — setting up the correct **view transformation** (camera's position and orientation).
- **5 marks** — correct **`MyTimer()`** function.

Note that marks will be deducted for bad coding style. If your program cannot be compiled and linked, you get 0 (zero) mark.

Good coding style. Comment your code adequately, use meaningful names for functions and variables, and indent your code properly. You must fill in your **name**, and **NUS User ID** in the **header comment**.

SUBMISSION

For this assignment, you need to **submit only** your completed **`main.cpp`**.

You must put it/them in a ZIP file and name your ZIP file ***nus-user-id*`_lab2.zip`**. For example, if your NUS User ID is **`e0123456`**, you should name your file **`e0123456_lab2.zip`**.

Note that you will be penalized for submitting non-required files.

Submit your ZIP file to **`Canvas > CS3241 > Assignments > Lab Assignment 2`**. Before the submission deadline, you may upload your ZIP file as many times as you want. **We will take only your latest submission**.

DEADLINE

Late submissions will NOT be accepted. The submission page will automatically close at the deadline.

——— End of Document ———

```
//============================================================
// STUDENT NAME:
// NUS User ID.:
// COMMENTS TO GRADER:
//
// ============================================================

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

#ifdef __APPLE__
#define GL_SILENCE_DEPRECATION
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

////////////////////////////////////////////////////////////////////////
// CONSTANTS
////////////////////////////////////////////////////////////////////////

#define PI                  3.14159265358979323846264338327950

#define PLANET_RADIUS       100.0

#define NUM_CARS            12      // Total number of cars.

#define CAR_LENGTH          32.0
#define CAR_WIDTH           16.0
#define CAR_HEIGHT          14.0

#define CAR_MIN_ANGLE_INCR  0.5     // Min degrees to rotate car around planet each
frame.
#define CAR_MAX_ANGLE_INCR  3.0     // Max degrees to rotate car around planet each
frame.

#define CAR_TOP_DIST        (PLANET_RADIUS + CAR_HEIGHT)  // Distance of the top of
a car from planet's center.

#define EYE_INIT_DIST       (3.0 * CAR_TOP_DIST)  // Initial distance of eye from
planet's center.
#define EYE_DIST_INCR       (0.1 * CAR_TOP_DIST)  // Distance increment when
changing eye's distance.
#define EYE_MIN_DIST        (1.5 * CAR_TOP_DIST)  // Min eye's distance from
planet's center.

#define EYE_LATITUDE_INCR   2.0     // Degree increment when changing eye's
latitude.
#define EYE_MIN_LATITUDE    -85.0   // Min eye's latitude (in degrees).
```

```
#define EYE_MAX_LATITUDE    85.0    // Max eye's latitude (in degrees).
#define EYE_LONGITUDE_INCR  2.0     // Degree increment when changing eye's
longitude.

#define CLIP_PLANE_DIST     (1.1 * CAR_TOP_DIST)  // Distance of near or far
clipping plane from planet's center.

#define VERT_FOV            45.0    // Vertical FOV (in degrees) of the perspective
camera.

#define DESIRED_FPS         60      // Approximate desired number of frames per
second.


// Planet's color.
const GLfloat planetColor[] = { 0.9, 0.6, 0.4 };

// Car tyre color.
const GLfloat tyreColor[] = { 0.2, 0.2, 0.2 };

// Material properties for all objects.
const GLfloat materialSpecular[] = { 1.0, 1.0, 1.0, 1.0 };
const GLfloat materialShininess[] = { 100.0 };
const GLfloat materialEmission[] = { 0.0, 0.0, 0.0, 1.0 };

// Light 0.
const GLfloat light0Ambient[] = { 0.1, 0.1, 0.1, 1.0 };
const GLfloat light0Diffuse[] = { 0.7, 0.7, 0.7, 1.0 };
const GLfloat light0Specular[] = { 0.9, 0.9, 0.9, 1.0 };
const GLfloat light0Position[] = { 1.0, 1.0, 1.0, 0.0 };

// Light 1.
const GLfloat light1Ambient[] = { 0.1, 0.1, 0.1, 1.0 };
const GLfloat light1Diffuse[] = { 0.7, 0.7, 0.7, 1.0 };
const GLfloat light1Specular[] = { 0.9, 0.9, 0.9, 1.0 };
const GLfloat light1Position[] = { -1.0, 0.0, -0.5, 0.0 };


////////////////////////////////////////////////////////////////////////
// GLOBAL VARIABLES
////////////////////////////////////////////////////////////////////////

// Define the cars.
typedef struct CarType {
    float bodyColor[3]; // RGB color of the car body.
    double angleIncr;   // Degrees to rotate car around planet each frame.
    double angularPos;  // Angular position of car around planet (in degrees).

    double xzAxis[2];   // A vector in the x-z plane. Contains the x and z
components respectively.
```

```
    double rotAngle;    // Rotation angle about the xzAxis[].
} CarType;

CarType car[ NUM_CARS ];    // Array of cars.


// Define eye position.
// Initial eye position is at [ 0, 0, EYE_INIT_DIST ] in the world frame,
// looking at the world origin.
// The up-vector is assumed to be [0, 1, 0].
double eyeLatitude = 0;
double eyeLongitude = 0;
double eyeDistance = EYE_INIT_DIST;


// Window's size.
int winWidth = 800;     // Window width in pixels.
int winHeight = 600;    // Window height in pixels.


// Others.
bool pauseAnimation = false;    // Freeze the cars iff true.
bool drawAxes = true;           // Draw world coordinate frame axes iff true.
bool drawWireframe = false;     // Draw polygons in wireframe if true, otherwise
polygons are filled.



////////////////////////////////////////////////////////////////////////
// Draw a car with its bottom on the z = 0 plane. The car is heading in the
// +x direction and its top facing the +z direction.
// The z-axis passes through the center of the car.
// The car body is drawn using the input color, and its tyres are drawn
// using the constant tyreColor.
// The car has size CAR_LENGTH x CAR_WIDTH x CAR_HEIGHT.
////////////////////////////////////////////////////////////////////////

void DrawOneCar( float bodyColor[3] )
{
    glColor3fv(bodyColor);

    //*****************************
    // WRITE YOUR CODE HERE.
    //
    // Draw the car body.
    //*****************************

    glColor3fv(tyreColor);

    //*****************************
```

```
    // WRITE YOUR CODE HERE.
    //
    // Draw the four tyres.
    //**************************
}



////////////////////////////////////////////////////////////////////////////
// Draw all the cars. Each is put correctly on its great circle.
////////////////////////////////////////////////////////////////////////////

void DrawAllCars( void )
{
    for ( int i = 0; i < NUM_CARS; i++ )
    {
        //**************************
        // WRITE YOUR CODE HERE.
        //**************************
    }
}



////////////////////////////////////////////////////////////////////////////
// Draw the x, y, z axes. Each is drawn with the input length.
// The x-axis is red, y-axis green, and z-axis blue.
////////////////////////////////////////////////////////////////////////////

void DrawAxes( double length )
{
    glPushAttrib( GL_ALL_ATTRIB_BITS );
    glDisable( GL_LIGHTING );
    glLineWidth( 3.0 );
    glBegin( GL_LINES );
        // x-axis.
        glColor3f( 1.0, 0.0, 0.0 );
        glVertex3d( 0.0, 0.0, 0.0 );
        glVertex3d( length, 0.0, 0.0 );
        // y-axis.
        glColor3f( 0.0, 1.0, 0.0 );
        glVertex3d( 0.0, 0.0, 0.0 );
        glVertex3d( 0.0, length, 0.0 );
        // z-axis.
        glColor3f( 0.0, 0.0, 1.0 );
        glVertex3d( 0.0, 0.0, 0.0 );
        glVertex3d( 0.0, 0.0, length );
    glEnd();
    glPopAttrib();
}
```

```
////////////////////////////////////////////////////////////////////////
// The display callback function.
////////////////////////////////////////////////////////////////////////

void MyDisplay( void )
{
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();

    //*********************************************************************
    // WRITE YOUR CODE HERE.
    //
    // Modify the following line of code to set up a perspective view
    // frustum using gluPerspective(). The near and far planes should be set
    // near the planet's surface, yet still do not clip off any part of the
    // planet and cars. The near and far planes should vary with the eye's
    // distance from the planet's center. You should make use of the value of
    // the predefined constant CLIP_PLANE_DIST to position your near and
    // far planes.
    //*********************************************************************
    gluPerspective( VERT_FOV, (double)winWidth / winHeight, 50.0, 600.0 );


    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();

    //*********************************************************************
    // WRITE YOUR CODE HERE.
    //
    // Modify the following line of code to set up the view transformation.
    // You may use the gluLookAt() function, but you can use other method.
    //*********************************************************************
    gluLookAt( 0.0, 0.0, eyeDistance, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0 );


    // Set world positions of the two lights.
    glLightfv( GL_LIGHT0, GL_POSITION, light0Position);
    glLightfv( GL_LIGHT1, GL_POSITION, light1Position);

    // Draw axes.
    if ( drawAxes ) DrawAxes( 2 * PLANET_RADIUS );

    // Draw planet.
    glColor3fv( planetColor );
    glutSolidSphere( PLANET_RADIUS, 72, 36 );
```

```
    // Draw the cars.
    DrawAllCars();

    glutSwapBuffers();
}



////////////////////////////////////////////////////////////////////////
// Update each car's angular position on the great circle by
// the angle increment.
////////////////////////////////////////////////////////////////////////

void UpdateCars( void )
{
    for ( int i = 0; i < NUM_CARS; i++ )
    {
        car[i].angularPos += car[i].angleIncr;
        if ( car[i].angularPos > 360.0 ) car[i].angularPos -= 360.0;
    }
    glutPostRedisplay();
}



////////////////////////////////////////////////////////////////////////
// Initializes each car with a random body color, a random rotation
// increment (speed), a random anglular position, and a random great circle.
////////////////////////////////////////////////////////////////////////

void InitCars( void )
{
    for ( int i = 0; i < NUM_CARS; i++ )
    {
        car[i].bodyColor[0] = (float)rand() / RAND_MAX;  // 0.0 to 1.0.
        car[i].bodyColor[1] = (float)rand() / RAND_MAX;  // 0.0 to 1.0.
        car[i].bodyColor[2] = (float)rand() / RAND_MAX;  // 0.0 to 1.0.

        car[i].angleIncr = (double)rand() / RAND_MAX *
                           ( CAR_MAX_ANGLE_INCR - CAR_MIN_ANGLE_INCR ) +
CAR_MIN_ANGLE_INCR;
                           // CAR_MIN_ANGLE_INCR to CAR_MAX_ANGLE_INCR.

        car[i].angularPos = (double)rand() / RAND_MAX * 360.0;      // 0.0 to
360.0.

        // The following 3 items defines a random great circle.
        car[i].xzAxis[0] = (double)rand() / RAND_MAX * 2.0 - 1.0;   // -1.0 to 1.0.
        car[i].xzAxis[1] = (double)rand() / RAND_MAX * 2.0 - 1.0;   // -1.0 to 1.0.
```

```
        car[i].rotAngle = (double)rand() / RAND_MAX * 360.0;        // 0.0 to
360.0.
    }
}



////////////////////////////////////////////////////////////////////////////////
// The timer callback function.
////////////////////////////////////////////////////////////////////////////////

void MyTimer( int v )
{
    if ( !pauseAnimation )
    {
        //****************************
        // WRITE YOUR CODE HERE.
        //****************************
    }
}



////////////////////////////////////////////////////////////////////////////////
// The keyboard callback function.
////////////////////////////////////////////////////////////////////////////////

void MyKeyboard( unsigned char key, int x, int y )
{
    switch ( key )
    {
        // Quit program.
        case 'q':
        case 'Q':
            exit(0);
            break;

        // Toggle between wireframe and filled polygons.
        case 'w':
        case 'W':
            drawWireframe = !drawWireframe;
            if ( drawWireframe )
                glPolygonMode( GL_FRONT_AND_BACK, GL_LINE );
            else
                glPolygonMode( GL_FRONT_AND_BACK, GL_FILL );
            glutPostRedisplay();
            break;

        // Toggle axes.
        case 'x':
```

```
        case 'X':
            drawAxes = !drawAxes;
            glutPostRedisplay();
            break;

         // Pause or resume animation.
        case 'p':
        case 'P':
            pauseAnimation = !pauseAnimation;
            if ( !pauseAnimation ) glutTimerFunc( 0, MyTimer, 0 );
            break;

       // Reset to initial view.
        case 'r':
        case 'R':
            eyeLatitude = 0.0;
            eyeLongitude = 0.0;
            eyeDistance = EYE_INIT_DIST;
            glutPostRedisplay();
            break;

    }
}



////////////////////////////////////////////////////////////////////////
// The special key callback function.
////////////////////////////////////////////////////////////////////////

void MySpecialKey( int key, int x, int y )
{
    switch ( key )
    {
        case GLUT_KEY_LEFT:
            eyeLongitude -= EYE_LONGITUDE_INCR;
            if ( eyeLongitude < -360.0 ) eyeLongitude += 360.0 ;
            glutPostRedisplay();
            break;

        case GLUT_KEY_RIGHT:
            eyeLongitude += EYE_LONGITUDE_INCR;
            if ( eyeLongitude > 360.0 ) eyeLongitude -= 360.0 ;
            glutPostRedisplay();
            break;

        case GLUT_KEY_DOWN:
            eyeLatitude -= EYE_LATITUDE_INCR;
            if ( eyeLatitude < EYE_MIN_LATITUDE ) eyeLatitude = EYE_MIN_LATITUDE;
            glutPostRedisplay();
```

```
            break;

        case GLUT_KEY_UP:
            eyeLatitude += EYE_LATITUDE_INCR;
            if ( eyeLatitude > EYE_MAX_LATITUDE ) eyeLatitude = EYE_MAX_LATITUDE;
            glutPostRedisplay();
            break;

        case GLUT_KEY_PAGE_UP:
            eyeDistance -= EYE_DIST_INCR;
            if ( eyeDistance < EYE_MIN_DIST ) eyeDistance = EYE_MIN_DIST;
            glutPostRedisplay();
            break;

        case GLUT_KEY_PAGE_DOWN:
            eyeDistance += EYE_DIST_INCR;
            glutPostRedisplay();
            break;
    }
}



////////////////////////////////////////////////////////////////////////
// The reshape callback function.
////////////////////////////////////////////////////////////////////////

void MyReshape( int w, int h )
{
    winWidth = w;
    winHeight = h;
    glViewport( 0, 0, w, h );
}



////////////////////////////////////////////////////////////////////////
// The init function. It initializes some OpenGL states.
////////////////////////////////////////////////////////////////////////

void MyInit( void )
{
    glClearColor( 0.0, 0.0, 0.0, 1.0 ); // Set black background color.
    glEnable( GL_DEPTH_TEST ); // Use depth-buffer for hidden surface removal.
    glShadeModel( GL_SMOOTH );

    //====================================================================
    // The rest of the code below sets up the lighting and
    // the material properties of all objects.
    // You can just ignore this part.
```

```
    //========================================================================

    // Set Light 0.
    glLightfv( GL_LIGHT0, GL_AMBIENT, light0Ambient );
    glLightfv( GL_LIGHT0, GL_DIFFUSE, light0Diffuse );
    glLightfv( GL_LIGHT0, GL_SPECULAR, light0Specular );
    glEnable( GL_LIGHT0 );

    // Set Light 1.
    glLightfv( GL_LIGHT1, GL_AMBIENT, light1Ambient );
    glLightfv( GL_LIGHT1, GL_DIFFUSE, light1Diffuse );
    glLightfv( GL_LIGHT1, GL_SPECULAR, light1Specular );
    glEnable( GL_LIGHT1 );

    glEnable( GL_LIGHTING );

    // Set some global light properties.
    GLfloat globalAmbient[] = { 0.1, 0.1, 0.1, 1.0 };
    glLightModelfv( GL_LIGHT_MODEL_AMBIENT, globalAmbient );
    glLightModeli( GL_LIGHT_MODEL_LOCAL_VIEWER, GL_FALSE );
    glLightModeli( GL_LIGHT_MODEL_TWO_SIDE, GL_FALSE );

    // Set the universal material properties.
    // The diffuse and ambient components can be changed using glColor*().
    glMaterialfv( GL_FRONT, GL_SPECULAR, materialSpecular );
    glMaterialfv( GL_FRONT, GL_SHININESS, materialShininess );
    glMaterialfv( GL_FRONT, GL_EMISSION, materialEmission );
    glColorMaterial( GL_FRONT, GL_AMBIENT_AND_DIFFUSE );
    glEnable( GL_COLOR_MATERIAL );

    glEnable( GL_NORMALIZE ); // Let OpenGL automatically renomarlize all normal
vectors.
}


static void WaitForEnterKeyBeforeExit(void)
{
    printf("Press Enter to exit.\n");
    fflush(stdin);
    getchar();
}



////////////////////////////////////////////////////////////////////////////
// The main function.
////////////////////////////////////////////////////////////////////////////

int main( int argc, char** argv )
{
```

```
    atexit(WaitForEnterKeyBeforeExit); // atexit() is declared in stdlib.h

    srand(27);  // set random seed

    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
    glutInitWindowSize( winWidth, winHeight );
    glutCreateWindow( "main" );

    MyInit();
    InitCars();

    // Register the callback functions.
    glutDisplayFunc( MyDisplay );
    glutReshapeFunc( MyReshape );
    glutKeyboardFunc( MyKeyboard );
    glutSpecialFunc( MySpecialKey );
    glutTimerFunc( 0, MyTimer, 0 );

    // Display user instructions in console window.
    printf( "Press LEFT ARROW to move eye left.\n" );
    printf( "Press RIGHT ARROW to move eye right.\n" );
    printf( "Press DOWN ARROW to move eye down.\n" );
    printf( "Press UP ARROW to move eye up.\n" );
    printf( "Press PAGE UP to move closer.\n" );
    printf( "Press PAGE DN to move further.\n" );
    printf( "Press 'P' to toggle car animation.\n" );
    printf( "Press 'W' to toggle wireframe.\n" );
    printf( "Press 'X' to toggle axes.\n" );
    printf( "Press 'R' to reset to initial view.\n" );
    printf( "Press 'Q' to quit.\n\n" );

    // Enter GLUT event loop.
    glutMainLoop();
    return 0;
}
```