1. (2 pts)

$$M_x = \begin{bmatrix} 0 & -4 & 3 \\ 4 & 0 & -2 \\ -3 & 2 & 0 \end{bmatrix}$$

2. (2 pts)

$$M_d = \begin{bmatrix} 4 & 6 & 8 \\ 6 & 9 & 12 \\ 8 & 12 & 16 \end{bmatrix}$$

3. (6 pts)

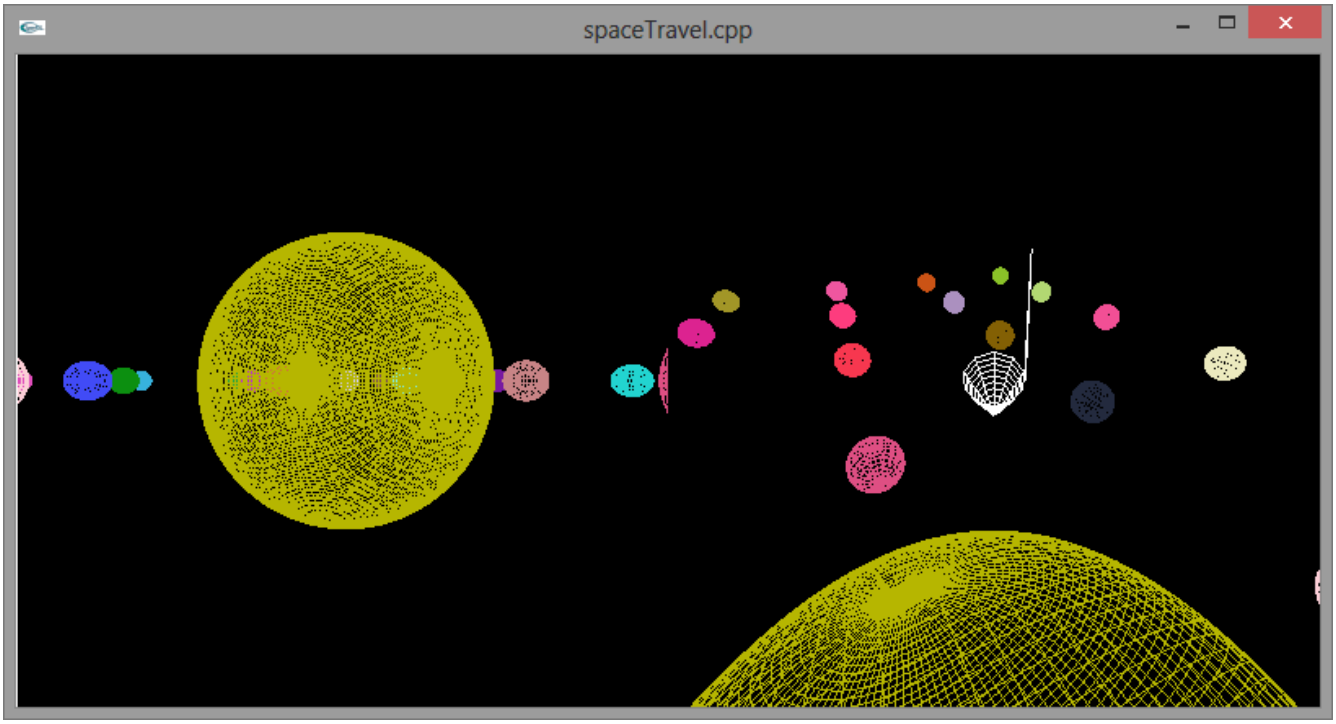
$$M = \begin{bmatrix} \cos 90 & 0 & \sin 90 \\ 0 & 1 & 0 \\ -\sin 90 & 0 & \cos 90 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos 90 & -\sin 90 \\ 0 & \sin 90 & \cos 90 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix}$$

$$|A - \lambda \cdot I| = \begin{vmatrix} -\lambda & 1 & 0 \\ 0 & -\lambda & -1 \\ -1 & 0 & -\lambda \end{vmatrix} = (-\lambda + 1)(\lambda^2 + \lambda + 1)$$

$$\lambda = -\frac{1}{2} \pm i\frac{\sqrt{3}}{2} \qquad \cos^{-1}\left(-\frac{1}{2}\right) = 120^o$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \begin{matrix} y = x \\ -z = y \\ -x = z \end{matrix} \qquad --> \qquad \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ \frac{-1}{\sqrt{3}} \end{bmatrix} (normalized)$$

4. (10 pts)




```
////////////////////////////////////////////////////////////////////////////////
// canvas.cpp
//
// HW5 Solution - Problem 4
// Guha - Problem 4.52 (page 155)
// Computer Graphics - Fall 2013
////////////////////////////////////////////////////////////////////////////////

#include <cstdlib>
#include <cmath>
#include <iostream>
#include <time.h>

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

#define PI 3.14159265

#define ROWS 8  // Number of rows of asteroids.
#define COLUMNS 5 // Number of columns of asteroids.
#define FILL_PROBABILITY 100 // Percentage probability that a particular row-column slot will
be
```

```
                              // filled with an asteroid. It should be an integer between 0 and
100.

using namespace std;

// Globals.
static long font = (long)GLUT_BITMAP_8_BY_13; // Font selection.
static int width, height; // Size of the OpenGL window.
static float angle = 0.0; // Angle of the spacecraft.
static float xVal = 0, zVal = 0; // Co-ordinates of the spacecraft.
static int isCollision = 0; // Is there collision between the spacecraft and an asteroid?
static bool goldenCollision = false; // Is there a collision between the spacecraft and the
golden asteroid?
static unsigned int spacecraft; // Display lists base index.
static float intensity = 1.0; // Intensity of the golden asteroid
static bool intensity_increasing = false; // Direction of the change in intensity

// Routine to draw a bitmap character string.
void writeBitmapString(void *font, char *string)
{
   char *c;

   for (c = string; *c != '\0'; c++) glutBitmapCharacter(font, *c);
}

// Asteroid class.
class Asteroid
{
public:
   Asteroid();
   Asteroid(float x, float y, float z, float r, unsigned char colorR,
                  unsigned char colorG, unsigned char colorB);
   float getCenterX() { return centerX; }
   float getCenterY() { return centerY; }
   float getCenterZ() { return centerZ; }
   float getRadius()  { return radius; }
   void setGolden();
   bool getGolden() { return isGolden; }
   void draw();

private:
   float centerX, centerY, centerZ, radius;
   bool isGolden;
   unsigned char color[3];
};

// Asteroid default constructor.
Asteroid::Asteroid()
{
   centerX = 0.0;
   centerY = 0.0;
   centerZ = 0.0;
   radius = 0.0; // Indicates no asteroid exists in the position.
   color[0] = 0;
```

```
   color[1] = 0;
   color[2] = 0;
   isGolden = 0;
}

// Asteroid constructor.
Asteroid::Asteroid(float x, float y, float z, float r, unsigned char colorR,
                                unsigned char colorG, unsigned char colorB)
{
   centerX = x;
   centerY = y;
   centerZ = z;
   radius = r;
   color[0] = colorR;
   color[1] = colorG;
   color[2] = colorB;
   isGolden = false;
}

void Asteroid::setGolden(){
       radius *= 5;
       color[0] = 255;
       color[1] = 255;
       color[2] = 0;
       isGolden = true;
}

// Function to draw asteroid.
void Asteroid::draw()
{
   if (radius > 0.0) // If asteroid exists.
   {
      glPushMatrix();
      glTranslatef(centerX, centerY, centerZ);
      // Adjust the color of the golden asteroid to account for intensity
         if (isGolden)
                glColor3ub((char)(intensity*color[0]),(char)(intensity*color[1]),(char)
(intensity*color[2]));
         else
                glColor3ubv(color);
      glutWireSphere(radius, (int)radius*6, (int)radius*6);
      glPopMatrix();
   }
}

Asteroid arrayAsteroids[ROWS][COLUMNS]; // Global array of asteroids.

// Initialization routine.
void setup(void)
{
   int i, j;

   spacecraft = glGenLists(1);
   glNewList(spacecraft, GL_COMPILE);
```

```
         glPushMatrix();
         glRotatef(180.0, 0.0, 1.0, 0.0); // To make the spacecraft point down the $z$-axis
initially.
         glColor3f (1.0, 1.0, 1.0);
      glutWireCone(5.0, 10.0, 10, 10);
         glPushMatrix();
         glRotatef(-90.0, 1.0, 0.0, 0.0); // To make the spacecraft point down the $z$-axis
initially.
         glTranslatef(5.0,0.0,0.0);
         glColor3f (1.0, 1.0, 1.0);
         glutWireCone(0.1,20,10,10);
         glPopMatrix();
         glPopMatrix();
   glEndList();

   // Initialize global arrayAsteroids.
   for (j=0; j<COLUMNS; j++)
      for (i=0; i<ROWS; i++)
            if (rand()%100 < FILL_PROBABILITY)
         // If rand()%100 >= FILL_PROBABILITY the default constructor asteroid remains in the
slot
               // which indicates that there is no asteroid there because the default's radius
is 0.
               {
                     // Position the asteroids depending on if there is an even or odd number
of columns
                     // so that the spacecraft faces the middle of the asteroid field.
                     if (COLUMNS%2) // Odd number of columns.
                  arrayAsteroids[i][j] = Asteroid( 30.0*(-COLUMNS/2 + j), 0.0, -40.0 - 30.0*i,
3.0,
                                                   rand()%256, rand()%256, rand()%256);
                     else // Even number of columns.
                        arrayAsteroids[i][j] = Asteroid( 15 + 30.0*(-COLUMNS/2 + j), 0.0, -40.0
- 30.0*i, 3.0,
                                                   rand()%256, rand()%256, rand()%256);
         // Set the golden asteroid
                     if((j==COLUMNS/2) && (i==ROWS/2)) arrayAsteroids[i][j].setGolden();
               }

   glEnable(GL_DEPTH_TEST);
   glClearColor (0.0, 0.0, 0.0, 0.0);
}

// Function to check if two spheres centered at (x1,y1,z1) and (x2,y2,z2) with
// radius r1 and r2 intersect.
int checkSpheresIntersection(float x1, float y1, float z1, float r1,
                                          float x2, float y2, float z2, float r2)
{
   return ( (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) + (z1-z2)*(z1-z2) <= (r1+r2)*(r1+r2) );
}

// Function to check if the spacecraft collides with an asteroid when the center of the base
// of the craft is at (x, 0, z) and it is aligned at an angle a to to the -z direction.
// Collision detection is approximate as instead of the spacecraft we use a bounding sphere.
```

```
int asteroidCraftCollision(float x, float z, float a)
{
   int i,j;

   // Check for collision with each asteroid.
   for (j=0; j<COLUMNS; j++)
      for (i=0; i<ROWS; i++)
               if (arrayAsteroids[i][j].getRadius() > 0 ) // If asteroid exists.
            if ( checkSpheresIntersection( x - 5 * sin( (PI/180.0) * a), 0.0,
                     z - 5 * cos( (PI/180.0) * a), 7.072,
                     arrayAsteroids[i][j].getCenterX(), arrayAsteroids[i][j].getCenterY(),
                     arrayAsteroids[i][j].getCenterZ(), arrayAsteroids[i][j].getRadius() ) )
            {
               if (arrayAsteroids[i][j].getGolden()) goldenCollision = true;
                     return 1;
            }
   return 0;
}

// Drawing routine.
void drawScene(void)
{
   int i, j;
   glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   // Alter intensity of golden asteroid
   if(intensity_increasing) intensity += 0.1;
   else intensity -= 0.1;
   if(intensity >= 1.0) intensity_increasing = false;
   else if (intensity <= 0.3) intensity_increasing = true;

   // Begin left viewport.
   glViewport (0, 0, width/2.0,  height);
   glLoadIdentity();

   // Write text in isolated (i.e., before gluLookAt) translate block.
   glPushMatrix();
   glColor3f(1.0, 0.0, 0.0);
   glRasterPos3f(-28.0, 25.0, -30.0);
   if (goldenCollision) writeBitmapString((void*)font, "You have found gold!");
   else if (isCollision) writeBitmapString((void*)font, "Cannot - will crash!");
   glPopMatrix();

   // Draw a vertical line on the left of the viewport to separate the two viewports
   glColor3f(1.0, 1.0, 1.0);
   glLineWidth(2.0);
   glBegin(GL_LINES);
      glVertex3f(-5.0, -5.0, -5.0);
      glVertex3f(-5.0, 5.0, -5.0);
   glEnd();
   glLineWidth(1.0);

   // Locate the camera at the tip of the cone and pointing in the direction of the cone.
   gluLookAt(xVal - 10 * sin( (PI/180.0) * angle),
```

```
                0.0,
                      zVal - 10 * cos( (PI/180.0) * angle),
                xVal - 11 * sin( (PI/180.0) * angle),
                      0.0,
             zVal - 11 * cos( (PI/180.0) * angle),
             0.0,
                      1.0,
                      0.0);

   // Draw all the asteroids in arrayAsteroids.
   for (j=0; j<COLUMNS; j++)
      for (i=0; i<ROWS; i++)
         arrayAsteroids[i][j].draw();

   // End left viewport.

   // Begin right viewport.
   glViewport(width/2.0, 0, width/2.0, height);
   glLoadIdentity();

   // Fixed camera.
   gluLookAt(arrayAsteroids[ROWS/2][COLUMNS/2].getCenterX(), 25, arrayAsteroids[ROWS/2]
[COLUMNS/2].getCenterZ(), xVal, 0.0, zVal, 0.0, 1.0, 0.0);

   // Draw all the asteroids in arrayAsteroids.
   for (j=0; j<COLUMNS; j++)
      for (i=0; i<ROWS; i++)
         arrayAsteroids[i][j].draw();

   // Draw spacecraft.
   glPushMatrix();
   glTranslatef(xVal, 0.0, zVal);
   glRotatef(angle, 0.0, 1.0, 0.0);
   glCallList(spacecraft);
   glPopMatrix();

   // End right viewport.

   glutSwapBuffers();
}

// OpenGL window reshape routine.
void resize(int w, int h)
{
   glViewport (0, 0, (GLsizei)w, (GLsizei)h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glFrustum(-5.0, 5.0, -5.0, 5.0, 5.0, 250.0);
   glMatrixMode(GL_MODELVIEW);

   // Pass the size of the OpenGL window.
   width = w;
   height = h;
}
```

```
// Keyboard input processing routine.
void keyInput(unsigned char key, int x, int y)
{
   switch (key)
   {
      case 27:
         exit(0);
         break;
      default:
         break;
   }
}

// Callback routine for non-ASCII key entry.
void specialKeyInput(int key, int x, int y)
{
   float tempxVal = xVal, tempzVal = zVal, tempAngle = angle;

   // Compute next position.
   if (key == GLUT_KEY_LEFT) tempAngle = angle + 5.0;
   if (key == GLUT_KEY_RIGHT) tempAngle = angle - 5.0;
   if( key == GLUT_KEY_UP)
   {
         tempxVal = xVal - sin(angle * PI/180.0);
         tempzVal = zVal - cos(angle * PI/180.0);
   }
   if( key == GLUT_KEY_DOWN)
   {
         tempxVal = xVal + sin(angle * PI/180.0);
         tempzVal = zVal + cos(angle * PI/180.0);
   }

   // Angle correction.
   if (tempAngle > 360.0) tempAngle -= 360.0;
   if (tempAngle < 0.0) tempAngle += 360.0;

  // Move spacecraft to next position only if there will not be collision with an asteroid.
   if (!asteroidCraftCollision(tempxVal, tempzVal, tempAngle) )
   {
      isCollision = 0;
      goldenCollision = false;
      xVal = tempxVal;
         zVal = tempzVal;
         angle = tempAngle;
   }
   else isCollision = 1;

   glutPostRedisplay();
}

// Routine to output interaction instructions to the C++ window.
void printInteraction(void)
{
```

```
   cout << "Interaction:" << endl;
   cout << "Press the left/right arrow keys to turn the craft." << endl
        << "Press the up/down arrow keys to move the craft." << endl;
}

// Main routine.
int main(int argc, char **argv)
{
   printInteraction();
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize(800, 400);
   glutInitWindowPosition(100, 100);
   glutCreateWindow("spaceTravel.cpp");
   setup();
   glutDisplayFunc(drawScene);
   glutReshapeFunc(resize);
   glutKeyboardFunc(keyInput);
   glutSpecialFunc(specialKeyInput);
   glutMainLoop();

   return 0;
}
```