Course Title: Introduction to Virtual Reality

Course Code: ENSF 545

Lab Number: 2

Instructor Name: Yaoping Hu

Student Names: Jennifer Patterson and James Thorne

Submission Date: October 10 2012

PART 1: EXERCISE 1

Step 13:

Upon commenting out the line, the rotating triangles are much lighter in color. The original triangles were a very dark red, blue and green, and the new ones were pastel.

Filenames:

./testProject/.caverc

./testProject/caveMain.cpp

./testProject/Debug

./testProject/Debug/BuildLog.htm

./testProject/Debug/caveMain.obj

./testProject/Debug/KNAVETest.exe

./testProject/Debug/KNAVETest.exe.embed.manifest

./testProject/Debug/KNAVETest.exe.embed.manifest.res

./testProject/Debug/KNAVETest.exe.intermediate.manifest

./testProject/Debug/mt.dep

./testProject/Debug/vc80.idb

./testProject/files.txt

./testProject/KNAVETest.dsp

./testProject/KNAVETest.ncb

./testProject/KNAVETest.plg

./testProject/KNAVETest.sln

./testProject/KNAVETest.suo

./testProject/KNAVETest.vcproj

./testProject/KNAVETest.vcproj.ENA10ARED.Administrator.user

./testProject/KNAVETest.vcproj.UC_CAMPUS.jdthorne.user

./testProject/Makefile

./testProject/Makefile.IRIX

./testProject/projectEx1CaveMain.cpp

./testProject/Release

./testProject/Release/BuildLog.htm

./testProject/Release/caveMain.obj

./testProject/Release/mt.dep

./testProject/Release/projectEx1.exe

./testProject/Release/projectEx1.exe.intermediate.manifest

./testProject/Release/projectEx1CaveMain.obj

./testProject/Release/testProject.exe

./testProject/Release/testProject.exe.intermediate.manifest

./testProject/Release/vc80.idb

Function draw(void):

```
void draw(void)
{
	//printf( "Drawing.\n" );

	glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

	glMatrixMode( GL_MODELVIEW );
	glPushMatrix();

	float headPos[3];
	float headVec[3];

	CAVEGetPosition( CAVE_HEAD, headPos );

	glPushMatrix();

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

	glDisable( GL_LIGHTING );

	//headLines
	CAVEGetVector( CAVE_HEAD_FRONT, headVec );
	glBegin( GL_LINES );
		glColor3f( 1,1,1 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
	glEnd();
```

```
CAVEGetVector( CAVE_HEAD_BACK, headVec );
glBegin( GL_LINES );
		glColor3f( 0.5,0.5,0.5 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_HEAD_LEFT, headVec );
glBegin( GL_LINES );
		glColor3f( 1,1,1 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_HEAD_RIGHT, headVec );
glBegin( GL_LINES );
		glColor3f( 0.5,0.5,0.5 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_HEAD_UP, headVec );
glBegin( GL_LINES );
		glColor3f( 1,1,1 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_HEAD_DOWN, headVec );
glBegin( GL_LINES );
		glColor3f( 0.5,0.5,0.5 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();

glPopMatrix();

//wand lines
CAVEGetPosition( CAVE_WAND, headPos );
glPushMatrix();
glTranslatef( headPos[0], headPos[1], headPos[2] );

CAVEGetVector( CAVE_WAND_FRONT, headVec );
glBegin( GL_LINES );
		glColor3f( 0.99,0.99,0.99 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_WAND_BACK, headVec );
glBegin( GL_LINES );
		glColor3f( 0.5,0.5,0.5 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_WAND_LEFT, headVec );
glBegin( GL_LINES );
		glColor3f( 0.99,0.99,0.99 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_WAND_RIGHT, headVec );
glBegin( GL_LINES );
		glColor3f( 0.5,0.5,0.5 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_WAND_UP, headVec );
glBegin( GL_LINES );
```

```
                glColor3f( 0.99,0.99,0.99 );
                glVertex3f( 0, 0, 0 );
                glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_WAND_DOWN, headVec );
glBegin( GL_LINES );
                glColor3f( 0.5,0.5,0.5 );
                glVertex3f( 0, 0, 0 );
                glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();

glPopMatrix();


CAVENavTransform();

glBegin( GL_QUADS );

glColor3f( 0.81, 0.81, 0.81 );

for( int i=-50; i<50;++i ){
                int shift = i%2;
                for( int j=-50; j<50;j+=2 ){

                                glVertex3f( j+shift+1, 0,  i   );
                                glVertex3f( j+shift,   0,  i   );
                                glVertex3f( j+shift,   0,  i+1 );
                                glVertex3f( j+shift+1, 0,  i+1 );

                }
}

glEnd();

//navigated head-lines
CAVEGetPosition( CAVE_HEAD_NAV, headPos );
glPushMatrix();
glTranslatef( headPos[0], headPos[1], headPos[2] );

//Begin test of head/wand vectors in navigated coords
CAVEGetVector( CAVE_HEAD_FRONT_NAV, headVec );
glBegin( GL_LINES );
                glColor3f( 1,1,1 );
                glVertex3f( 0, 0, 0 );
                glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_HEAD_BACK_NAV, headVec );
glBegin( GL_LINES );
                glColor3f( 0.5,0.5,0.5 );
                glVertex3f( 0, 0, 0 );
                glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_HEAD_LEFT_NAV, headVec );
glBegin( GL_LINES );
                glColor3f( 1,1,1 );
                glVertex3f( 0, 0, 0 );
                glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_HEAD_RIGHT_NAV, headVec );
glBegin( GL_LINES );
                glColor3f( 0.5,0.5,0.5 );
                glVertex3f( 0, 0, 0 );
                glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
```

```
CAVEGetVector( CAVE_HEAD_UP_NAV, headVec );
glBegin( GL_LINES );
		glColor3f( 1,1,1 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_HEAD_DOWN_NAV, headVec );
glBegin( GL_LINES );
		glColor3f( 0.5,0.5,0.5 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();

glPopMatrix();

//wand lines
CAVEGetPosition( CAVE_WAND_NAV, headPos );
glPushMatrix();
glTranslatef( headPos[0], headPos[1], headPos[2] );

CAVEGetVector( CAVE_WAND_FRONT_NAV, headVec );
glBegin( GL_LINES );
		glColor3f( 0.99,0.99,0.99 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_WAND_BACK_NAV, headVec );
glBegin( GL_LINES );
		glColor3f( 0.5,0.5,0.5 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_WAND_LEFT_NAV, headVec );
glBegin( GL_LINES );
		glColor3f( 0.99,0.99,0.99 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_WAND_RIGHT_NAV, headVec );
glBegin( GL_LINES );
		glColor3f( 0.5,0.5,0.5 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_WAND_UP_NAV, headVec );
glBegin( GL_LINES );
		glColor3f( 0.99,0.99,0.99 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();
CAVEGetVector( CAVE_WAND_DOWN_NAV, headVec );
glBegin( GL_LINES );
		glColor3f( 0.5,0.5,0.5 );
		glVertex3f( 0, 0, 0 );
		glVertex3f(headVec[0], headVec[1], headVec[2] );
glEnd();


glPopMatrix();

// Exercise 1, Step 13, Part 1
// glEnable( GL_LIGHTING );
//end nav vectors test
```

```
//triangle 1
glPushMatrix();

glTranslatef( 0, 0, -2.0 );

glRotatef( rotateAmt, 0, 1, 0 );

glBegin( GL_TRIANGLES );
        glColor3f( 0.7, 0.7, 0.7 );
        glVertex3f( -1.0, -0.5, 0.0 );
        glVertex3f(  0.0,  1.0, 0.0 );
        glVertex3f(  1.0, -0.5, 0.0 );
glEnd();

glPopMatrix();

//triangle 2
glPushMatrix();

glTranslatef( -6, 0, 0.0 );

// Exercise 1, Step 13, Part 2
/* The triangle below has been deleted:
glBegin( GL_TRIANGLES );
        glColor3f( 0.99, 0.7, 0.7 );
        glVertex3f( -1.0, -0.5, 0.0 );
        glVertex3f(  0.0,  1.0, 0.0 );
        glVertex3f(  1.0, -0.5, 0.0 );
glEnd();
*/

// The square now rotates by the x-axis
glRotatef( rotateAmt, 1, 0, 0 );

// The square is "2 units" in size as it is 2 units x 2 units.
glBegin(GL_QUADS);
        glColor3f(1.0f, 0.5f, 0.0f);
        glVertex3f(-1.0f, -1.0f, 0.0f);
        glVertex3f(+1.0f, -1.0f, 0.0f);
        glVertex3f(+1.0f, +1.0f, 0.0f);
        glVertex3f(-1.0f, +1.0f, 0.0f);
glEnd();

glPopMatrix();

//triangle 3
glPushMatrix();

glTranslatef( 6, 0, 0.0 );

// Exercise 1, Step 13, Part 3
// The triangle already rotates about the y-axis
glRotatef( rotateAmt, 0, 1, 0 );

// The verticies have been colored below
glBegin( GL_TRIANGLES );
        glColor3f(1.0f, 0.0f, 0.0f);
        glVertex3f( -1.0, -0.5, 0.0 );

        glColor3f(0.0f, 1.0f, 0.0f);
        glVertex3f(  0.0,  1.0, 0.0 );

        glColor3f(0.0f, 0.0f, 1.0f);
        glVertex3f(  1.0, -0.5, 0.0 );
```

```
        glEnd();

        glPopMatrix();

        //triangle 4
        glPushMatrix();

        glTranslatef( 0, 0, -6.0 );

        // Exercise 1, Step 13, Part 4

        /* The triangle has been deleted:
        glBegin( GL_TRIANGLES );
                glColor3f( 0.7, 0.7, 0.99 );
                glVertex3f( -1.0, -0.5, 0.0 );
                glVertex3f(  0.0,  1.0, 0.0 );
                glVertex3f(  1.0, -0.5, 0.0 );
        glEnd();
        */
        static float pyramidRotation = 0.0f;

        // The pyramid rotates by 0.2f per frame
        pyramidRotation += 0.2f;
        glRotatef(pyramidRotation, 0, 0, 1);

        float halfSqrt3 = 1.732f / 2.0f;

        // The new pyramid's geometry
        glBegin(GL_TRIANGLES);
                // "Bottom" face
                glColor3f(1.0f, 0.0f, 0.0f);
                glVertex3f(-1.0f, -halfSqrt3, -halfSqrt3);
                glVertex3f(0.0f, +halfSqrt3, -halfSqrt3);
                glVertex3f(+1.0f, -halfSqrt3, -halfSqrt3);

                // "Front" face
                glColor3f(0.0f, 1.0f, 0.0f);
                glVertex3f(-1.0f, -halfSqrt3, -halfSqrt3);
                glVertex3f(+1.0f, -halfSqrt3, -halfSqrt3);
                glVertex3f(0.0f, 0.0f, +halfSqrt3);

                // "Left" face
                glColor3f(0.0f, 0.0f, 1.0f);
                glVertex3f(+1.0f, -halfSqrt3, -halfSqrt3);
                glVertex3f(0.0f, +halfSqrt3, -halfSqrt3);
                glVertex3f(0.0f, 0.0f, +halfSqrt3);

                // "Right" face
                glColor3f(1.0f, 1.0f, 0.0f);
                glVertex3f(0.0f, +halfSqrt3, -halfSqrt3);
                glVertex3f(-1.0f, -halfSqrt3, -halfSqrt3);
                glVertex3f(0.0f, 0.0f, +halfSqrt3);

        glEnd();

        glPopMatrix();

        glPopMatrix();

}
```

PART 2: EXERCISE 2

Step 5) The quality of the stereoscopic view on workstation 13, using the 3D shutter glasses was the best. The red cube has no blurry lines and appeared sharp and realistic. However, the shutter glasses make the rest of the room look overly saturated and flickering.

The stereoscopic view of workstation 11, using the polarized glasses, did not as accurately create a 3D image. Lines were blurry and the vertical field of view was substantially limited. When not looking directly at the screen, the polarized lenses did not cause any significant disruption to our vision.

In both cases, the red circle was only visible to the right eye, breaking the immersion of the scene.

Overall, the 3D shutter glasses produced the best 3D experience.

PART 2: EXERCISE 3

Step 9:

1)

- Rendering Of Materials of Teapot: diffuseColor, diffuseColorRed, diffuseColorPurple, specularColor, drawTeapot(), createTeapotDL(), drawTeapotVBO(), initGL()
- Displaying Stereoscopic View: stereo, drawInStereo(), drawEye(), drawEyeLookAt(),
- Interfacing With a Keyboard: keyboardCB(), initGLUT(),

2)

- Rendering Of Materials of Teapot: initGL(), drawTeapot(), drawTeapotVBO(), createTeapotDL()
- Displaying Stereoscopic View: initSharedMem()
- Interfacing With a Keyboard: initGLUT()

3)

- Initial Color of Teapot: diffuseColorRed = {0.929524, 0.1, 0.178823}, specularColor = {1.0, 0.980392, 0.549020, 1.0}

- Position and Look-at Direction of the Camera: Initial Position = {0, 0, 10}, Look-at Position = {0, 0, 0}
- Focal Length Of the Camera: camera.focalLength = 14
- Initial Distance Between Two Eyes: camera.eyeSep = .23
- Initial Mode of Stereoscopic View: stereoMethod = false (2 Cameras)
- Total Number of Modes of Stereoscopic View: 2 Modes: 2 frustrums, or 2 cameras.

4)

2 Cameras:

- Positions of the two eyes are calculated.
- For each eye
    - Sets up the frustrum moved by the eye separation.
    - Set up the camera to look from the eye position to the focal point moved by the eye position.
    - Draw scene
- Swap OpenGL Buffers

2 Frustrums:

- Sets up an OpenGl frustrum moved by each eye.
- Translates frustrum by the half distance between the two eyes.

Pros and Cons:

- 2 Frustrums method is simpler to calculate, but difficult to understand.
- 2 Cameras method is easier to understand, but more complicated.
- 2 Cameras method looks more realisticly 3D.

Filenames:

./teapotLab

./teapotLab/Debug

./teapotLab/Debug/BuildLog.htm

./teapotLab/Debug/main.obj

./teapotLab/Debug/mt.dep

./teapotLab/Debug/teapotLab.exe

./teapotLab/Debug/teapotLab.exe.intermediate.manifest

./teapotLab/Debug/teapotLab.ilk

./teapotLab/Debug/teapotLab.pdb

./teapotLab/Debug/Timer.obj

./teapotLab/Debug/vc80.idb

./teapotLab/Debug/vc80.pdb

./teapotLab/Debug/vc90.idb

./teapotLab/Debug/vc90.pdb

./teapotLab/main.cpp

./teapotLab/Release

./teapotLab/Release/BuildLog.htm

./teapotLab/Release/main.obj

./teapotLab/Release/mt.dep

./teapotLab/Release/projectEx3.exe

./teapotLab/Release/projectEx3.exe.intermediate.manifest

./teapotLab/Release/projectEx3.pdb

./teapotLab/Release/Timer.obj

./teapotLab/Release/vc90.idb

./teapotLab/Release/vc90.pdb

./teapotLab/teapot.h

./teapotLab/teapotLab.ncb

./teapotLab/teapotLab.sln

./teapotLab/teapotLab.suo

./teapotLab/teapotLab.vcproj

./teapotLab/teapotLab.vcproj.UC_CAMPUS.jdthorne.user

./teapotLab/teapotLab.vcproj.UC_CAMPUS.jolin.user

./teapotLab/Timer.cxx

./teapotLab/Timer.h

./teapotLab/UpgradeLog.XML

./teapotLab/VTune

./teapotLab/VTune/teapotLab.vpj

./teapotLab/VTune/teapotLab.vws

Functions:

```
////////////////////////////////////////////////////////////////////////////////
int main(int argc, char **argv)
{
   // initialize global variables
   initSharedMem();

            // set initial color of teapot
            for (int i=0; i<colorIndex; i++){
                     diffuseColor[i]=diffuseColorRed[i];
            }

            // parameters for 3d stereoscopy
            camera.aperture = foview;// field of view

             camera.focalLength = 14;
   camera.eyeSep = camera.focalLength /60; // separation of the eyes

   //camera.focalLength = 70;
   //camera.eyeSep = camera.focalLength /300; // separation of the eyes

            // position of the camera
            camera.vp.x = 0;
   camera.vp.y = 0;
   camera.vp.z = 10;
   //view direction vector
   camera.vd.x = 0;
   camera.vd.y = 0;
   camera.vd.z = -1;
   //view up vector
   camera.vu.x = 0;
   camera.vu.y = 1;
   camera.vu.z = 0;

   // init GLUT and GL
   initGLUT(argc, argv);
   initGL(); //initialize opengl light and camera
            #ifdef HAPTIC
                     initHL(); //initialize haptic device
            #endif

   // compile a display list of teapot mesh
   // see detail in createTeapotDL()
   listId = createTeapotDL();

   // the last GLUT call (LOOP)
   // window will be shown and display callback is triggered by events
   // NOTE: this call never return main().
```

```
    glutMainLoop(); /* Start GLUT event-processing loop */

    return 0;
}

void keyboardCB(unsigned char key, int x, int y)
{
    switch(key)
    {
    case 27: // ESCAPE
        clearSharedMem();
        exit(0);
        break;

    case 'd': // switch rendering modes (fill -> wire -> point)
    case 'D':
                            /*
        drawMode = ++drawMode % 3;
        if(drawMode == 0)        // fill mode
        {
            glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
            glEnable(GL_DEPTH_TEST);
            glEnable(GL_CULL_FACE);
        }
        else if(drawMode == 1)  // wireframe mode
        {
            glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
            glDisable(GL_DEPTH_TEST);
            glDisable(GL_CULL_FACE);
        }
        else                    // point mode
        {
            glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);
            glDisable(GL_DEPTH_TEST);
            glDisable(GL_CULL_FACE);
        }
                            */
                            toggleEyeDistance();
        break;

    case ' ': // switch rendering modes (fill -> wire -> point)
        stereoMethod=!stereoMethod;
        break;
                case 'c':
                case 'C':
                            touched=!touched;
                            toggleTeapotColor();
                            // color();
                            break;

    default:
        ;
    }
    glutPostRedisplay();
}
void toggleTeapotColor(){

                static int currentColor = 0;
                currentColor = (currentColor + 1) % 3;

                const float* targetDiffuseColor = NULL;
                const float* targetSpecularColor = NULL;

                switch (currentColor)
                {
                            case 0:
                            {
                                        targetDiffuseColor = diffuseColorRed;
                                        targetSpecularColor = specularColorRed;
                                        break;
                            }
                            case 1:
```

```
                {
                        targetDiffuseColor = diffuseColorLab;
                        targetSpecularColor = specularColorLab;
                        break;
                }
                case 2:
                {
                        targetDiffuseColor = diffuseColorCustom;
                        targetSpecularColor = specularColorCustom;
                        break;
                }
        }

        for (int i = 0; i < 3; i++){
                diffuseColor[i] = targetDiffuseColor[i];
        }
        for (int i = 0; i < 4; i++){
                specularColor[i] = targetSpecularColor[i];
        }
}
void toggleEyeDistance()
{
        static int currentDistanceIndex = 0;
        currentDistanceIndex = (currentDistanceIndex + 1) % 3;

        switch (currentDistanceIndex)
        {
                case 0:
                {
                        camera.eyeSep = 0.23333;
                        break;
                }
                case 1:
                {
                        camera.eyeSep = 0.40;
                        break;
                }
                case 2:
                {
                        camera.eyeSep = 0.30;
                        break;
                }
        }
}
```