

Using Haptics to Simulate Medical Diagnoses

Christine Cabrera, Advisor: Dr. Tom Stewart

Abstract—This project utilizes the SensAble Technologies Phantom Omni haptic device to research the meaning of “touch” as a medical diagnosis. Providing a simulation that includes correct haptic feedback can offer a significantly more realistic environment than graphics alone. C++ and OpenGL software was written to create a simple virtual environment containing multiple objects with varying levels of resistance. The environment created demonstrates the collaboration of both the graphical and haptic interfaces that allow an operator to “feel” what is seen on the computer screen. This research is meant to create awareness about haptics as a powerful simulation tool, and lay foundation work from which future haptic projects can build.

Index Terms— medical simulation, force-feedback, haptic device, haptics technology.

I. Introduction

The market for medical simulators is growing dramatically as an increase in technology is allowing these devices to come to life. Creating virtual environments for the medical industry can provide cost-effective training and the opportunity for repetitive learning. This project moves beyond surgical tools and instruments, and utilizes a haptic device to research the meaning of “touch” as a diagnosis. What lies under the skin is visually unknown and applying pressure to the area (and knowing what it should feel like) results in the initial diagnosis.

A. What is a Haptic Device?

A haptic device is an electronic machine that creates a virtual three-dimensional environment allowing the operator to feel and touch virtual objects. The device can resist movement in specific locations so that the operator can feel where virtual objects have been placed. The device can mimic various feelings of touch from a slippery, malleable substance to one that is rock solid. This feeling, or touch, is also referred to as force-feedback from the system. Terms such as haptic device and haptics cursor are interchangeable in this research paper.

B. Future Applications for Haptics

This research focuses mainly on applications for haptics in the medical industry, specifically in training and simulation environments. Providing

proper and realistic feedback in a virtual simulation can teach an inexperienced doctor to become sensitive to the significance of various levels of pressure. In collaboration with complex graphical environments used for surgical simulations, haptics can replicate the texture of organs, tissues, and bones, for example. These simulations can also be extended to training individuals in all levels of medical professions, including doctors, nurses, medical technicians, and military medical units.

II. System Description

To create the virtual environment, software was written to the haptic device using C++ commands and functions from the OpenHaptics Toolkit reference manual.

A. Initial Research

The OpenHaptics Toolkit reference manual and sample software programs available were used during the initial research for operating the device, understanding device positions, orientation, and physical characteristics of objects.

1) Phase 1 - Software

Referring to Fig. 1 below, the first phase of this project development relied heavily on creating a virtual environment solely based on the haptic interface.

2) Phase 2 - Graphics

The second phase involved writing graphics to the objects created. An open source imaging software called OpenGL was used to create the graphics.

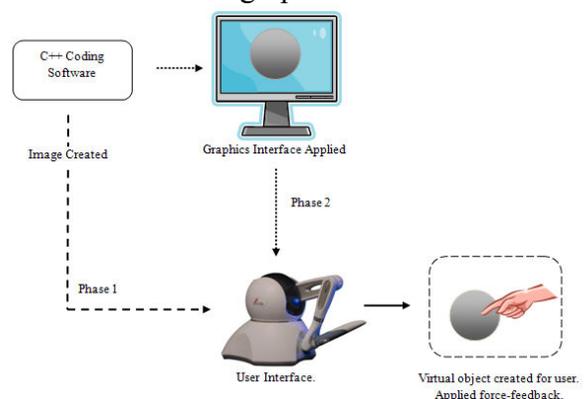


Fig. 1. High level overall system block diagram.

III. Project Development – Phase 1

A. Initial Two-Sphere Design

For the initial design task, it was proposed to create two spheres. One sphere would be made up of a very stiff surface inside a larger, more malleable sphere. The intent was for the operator to “push into” the first sphere and feel the harder surface of the second sphere, mimicking the application of pressure to an area and feeling its force-feedback.

After numerous attempts, the cursor could not locate the inside sphere. This may have been due to the code defaulting the spheres as a “solid” object and thus didn’t know how to compute an object inside the other. This will require some additional research to correct.

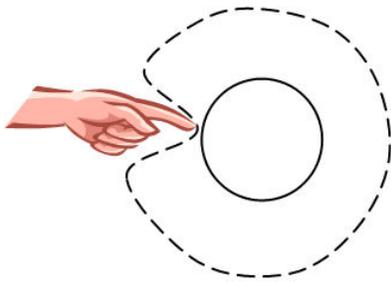


Fig. 2. Image of the two-sphere design concept.

B. Forces by Position Parameters

A second design idea was to map out the position of the cursor and impose forces at specific positions. For example, if the cursor’s X positions are between say, 15 and 20 points, then apply Force = $K * X$, where K is equal to a force constant. Also, the force applied in this design increases (or pushes back) as the cursor passes through the X frames.

This design worked well when it only had two reference planes. The next step was to create a stiffer force for the user to feel once they have passed through that first layer. However, after introducing another parameter to create a stiffer force, say at $20 < X < 25$ some instability was created with the device causing it to create a “buzzing” sound. The instability lessened if the stiffer force overlapped, for example at $18 < X < 23$.

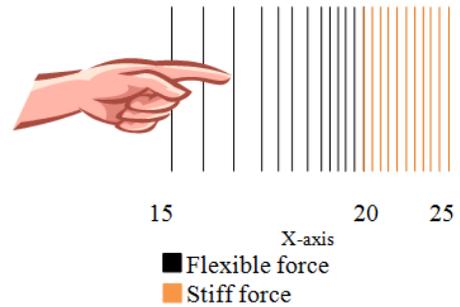


Fig. 3. Image of position parameter with end-to-end forces. Created instability on device.

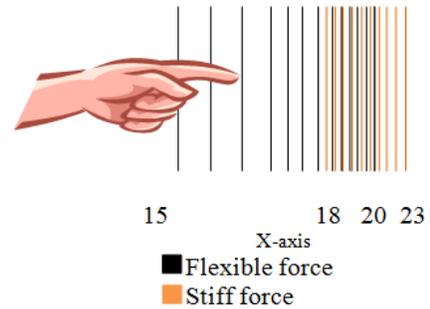


Fig. 4. Image of position parameter with overlapping forces. Created more stability on device.

From experimentation came two additional design questions: 1) Can objects be created solely on specifying certain X,Y,Z values? and 2) If not, can this be used as a layer over OpenGL cubes and spheres?

Question 1 – The expected answer is, yes, of course. Mathematically, one can create infinite objects with X,Y,Z values. However, the most that was proved here was an attempt at creating a cube. Basically, it consisted of setting forces over all three dimensions: Set a force for X values (length), a force for Y values (width), and force for Z values (height). While this did work correctly, the instability was too much to work with and the cube created through OpenGL created a much more well-defined shape.

Question 2 – This brings about the second question. If the instability is caused by created multiple layers, why not create one layer on top of already existing spheres or cubes? This became part of a new design. The first layer was a force set on specific X values, say for $20 < X < 40$. Then a sphere was created with its center position just behind the layer, say at $X > (20 - R)$ where R is the radius of the sphere. Initially, whenever the cursor reached the surface of the sphere, the forces of the

first layer would disappear. After numerous attempts of readjusting the programming, the correction needed was to set the force of the sphere equal to its force plus that of the layer. Originally it was only set to equal the forces of the sphere only. Only haptics were rendered, no graphics were programmed. An image of the haptics created is shown in Fig. 5 (a).

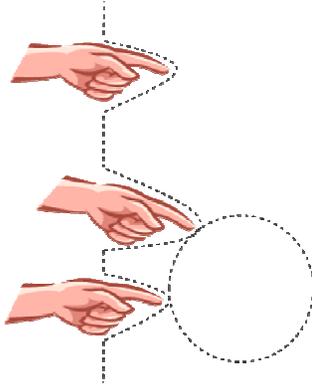


Fig. 5. (a) Image of haptics created with sphere drawn behind a position-parameter layer.

IV. Project Development – Phase 2

A. Initial Graphics Design

The idea was to mimic the design shown in Figure 5 and apply graphics. The first step was to create a graphical sphere and locate it directly inside of a graphical cube. However the problems found in the two-sphere design were brought up again at this point. The haptic device can push through the layer of the cube but no matter how hard it is pushed or how close the sphere is to the surface of the cube, the cursor could never feel the surface of the sphere. The conclusion made was that the programming creates solid cubes and solid spheres. Pressing through the cube will only allow the user to feel forces from the cube, and nothing else. Further research will need to be done to correct this issue.

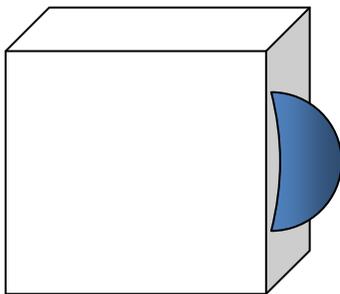


Fig. 6. Image of graphics created in OpenGL with sphere drawn behind a three-dimensional plane.

B. Final Graphics Design

The second attempt for creating graphics was to create multiple cubes, possibly in a row that contained various instances of friction and stiffness. Friction and stiffness are important parameters that help describe the surface of an object to the operator. Friction determines whether something is smooth like a tendon, or rough like a bone. Stiffness also determines the malleability factor – how dense or flimsy an object might be. The final design consisted of three cubes in a row, imitating a rectangular surface. The cube on the left contained a malleable surface, and the cube on the right had a variation in friction and stiffness. There were also two spheres present in the scene, one to show the operator what it looked like and how it felt, and the second was hidden on the surface of the middle cube mimicking a “lump” that the operator would have to locate. This design was shown in the Bradley University Student Exposition.

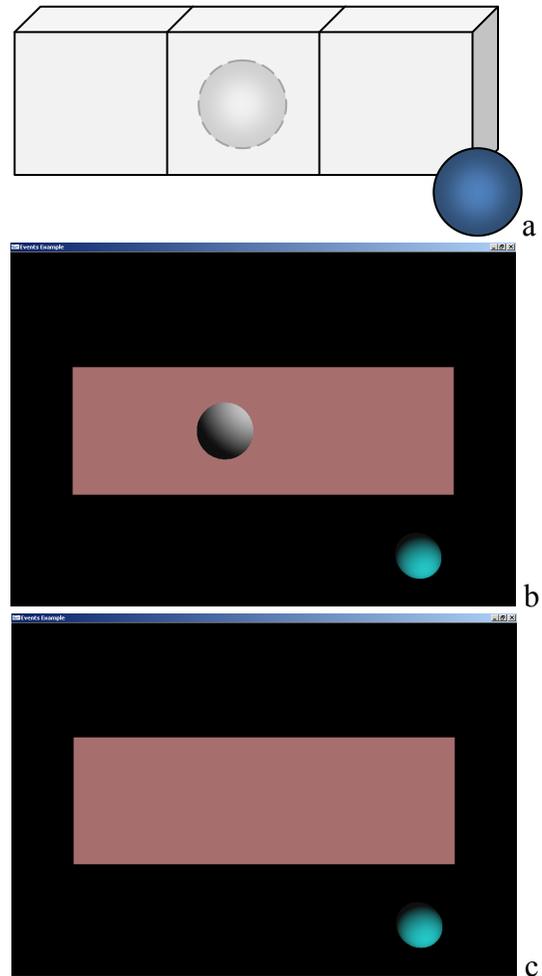


Fig. 7. (a) Graphical representation of final design. (b) Computer screenshot of design with center sphere, “lump” shown. (c) Computer screenshot with sphere, “lump” hidden.

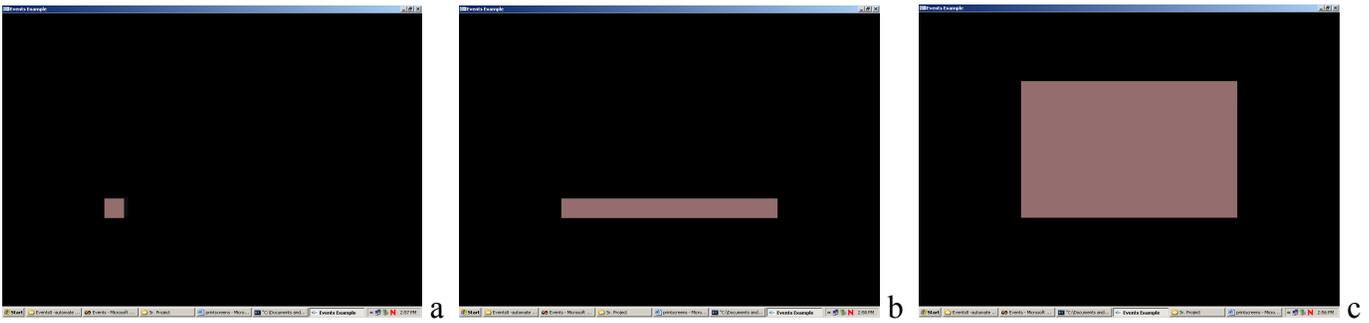


Fig. 8. (a) Screenshot creating one cube, voxel. (b) Screenshot after writing a loop to create back-to-back cubes, after specifying number of columns. (c) Screenshot after writing a nested loop in (b) to add rows of cubes, ultimately creating a plane.

C. Experimental Graphics Design

While not completed, this design was a continuation of the final graphics design. If a cube represented a voxel, or pixel, then one can ideally create hundreds or thousands of small cubes to create a large environment consisting of hundreds of various forces. Figures 10-12 show a progression of looped programming to instantly create a plane of sixty cubes.

V. Conclusions

This project was developed to initiate research in the area of haptics. While much of the experimentation was trial and error, this paper begins to show a procedure for starting from scratch to creating a simple haptic scene. The research will provide a foundation from which another student can learn and continue to develop more complex environments. Programming in C++ was used throughout a large majority of this project. Located in the appendices is a tutorial to start a program and some notable code from various designs.

A. Suggestions for Future Projects

Research more into the area of OpenGL. It may be able to provide opportunity for creating better, more complex graphics than in C++ alone.

VI. References

[1] Cohen, Georgy. Visualization Wall Grand Opening. Tufts University School of Engineering. 2008. 31 Mar. 2008
 <http://engineering.tufts.edu/1181647322330/Engineering-Page-eng2w_1202727607540.html>.

[2] Kesavadas, T, and Amrita Chanda. "Physically-Based Modeling through a Dynamic Atomic Unit Approach for Haptic Rendering: Towards Non-

Linear, Viscoelastic, Anisotropic Behavior." Virtual Reality Laboratory. University At Buffalo. 27 Feb. 2008

<http://www.vrlab.buffalo.edu/projects_group_medical/atomic_unit/atomic_nonlinear.html>.

[3] "Phantom Omni Haptic Device." Sensable Technologies. 31 Mar. 2008

<<http://www.sensable.com/haptic-phantom-omni.htm>>.

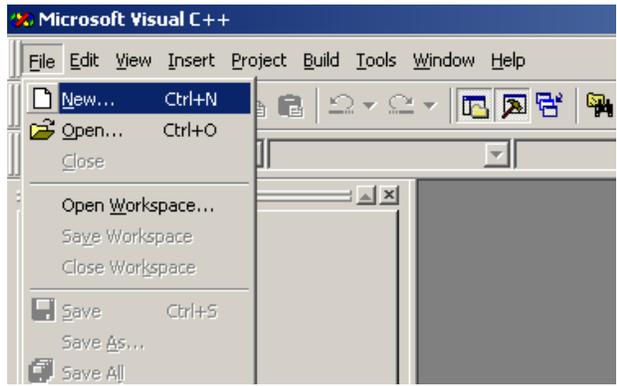
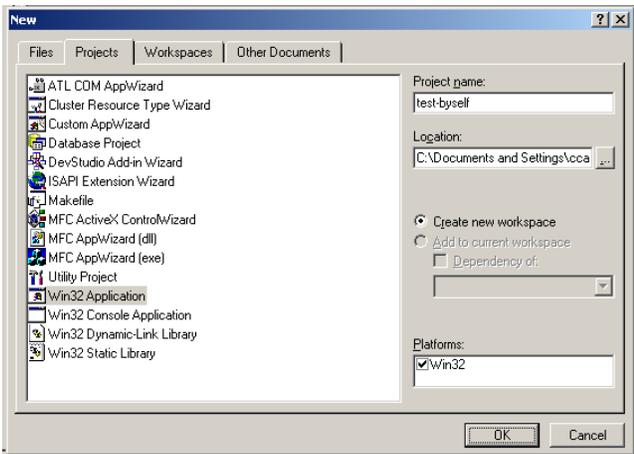
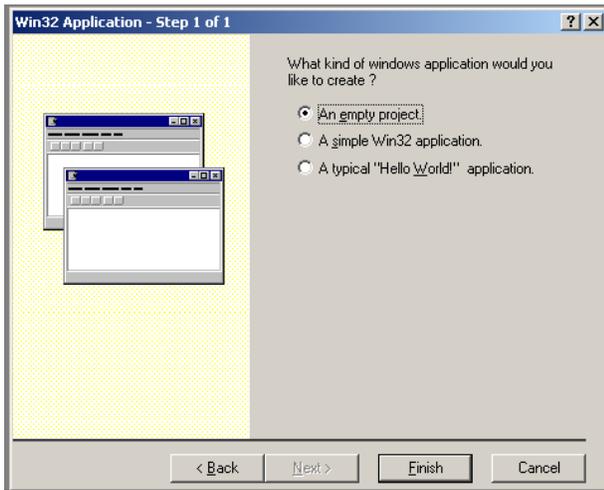
[4] Riezenman, Michael J. "Haptics Takes Hold." The Institute (2008). 30 Mar. 2008

<http://www.theinstitute.ieee.org/portal/site/tionline/menuitem.130a3558587d56e8fb2275875bac26c8/index.jsp?&pName=institute_level1_article&TheCat=2201&article=tionline/legacy/inst2008/mar08/featuretechnology.xml&>.

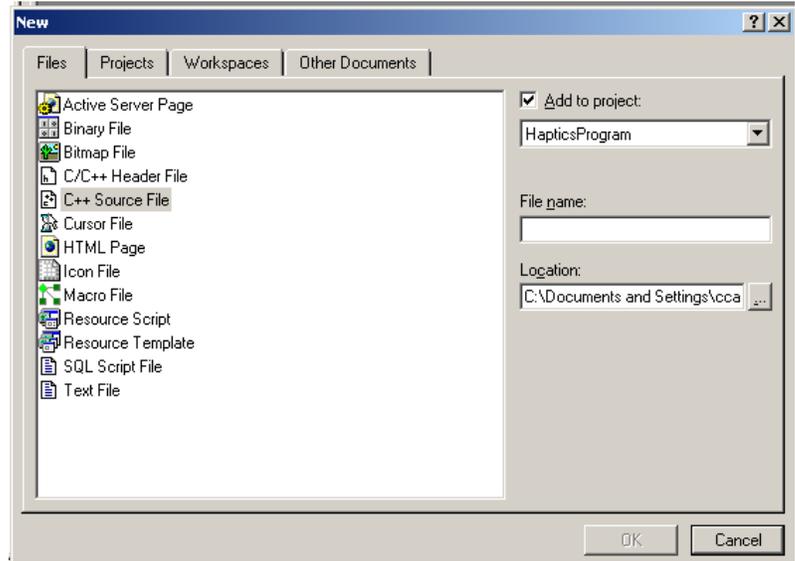
Appendix A

Tutorial Visual C++ Code to Draw a Simple Sphere*

*Tutorial Created off of Sample Code Provided in FrictionlessSphere.cpp

<p>Open Microsoft Visual C++. Click File, New.</p>	 <p>The screenshot shows the Microsoft Visual C++ application window. The 'File' menu is open, and the 'New...' option is highlighted. Other visible options include 'Open...', 'Close', 'Open Workspace...', 'Save Workspace', 'Close Workspace', 'Save', 'Save As...', and 'Save All'.</p>
<p>Choose Win32 Application. Name your project.</p>	 <p>The screenshot shows the 'New' dialog box in Visual C++. The 'Projects' tab is selected, and 'Win32 Application' is highlighted in the list. The 'Project name' field contains 'test-byself', and the 'Location' is 'C:\Documents and Settings\lcca...'. The 'Platforms' section has 'Win32' selected.</p>
<p>Choose "An empty project." Click Finish.</p>	 <p>The screenshot shows the 'Win32 Application - Step 1 of 1' wizard. The question is 'What kind of windows application would you like to create?'. The options are: 'An empty project.' (selected), 'A simple Win32 application.', and 'A typical "Hello World!" application.'. The 'Finish' button is visible at the bottom.</p>

Click File, New.
 Choose C++ Source File.
 Give the file a name. This file will automatically be added to your project.
 Click OK.



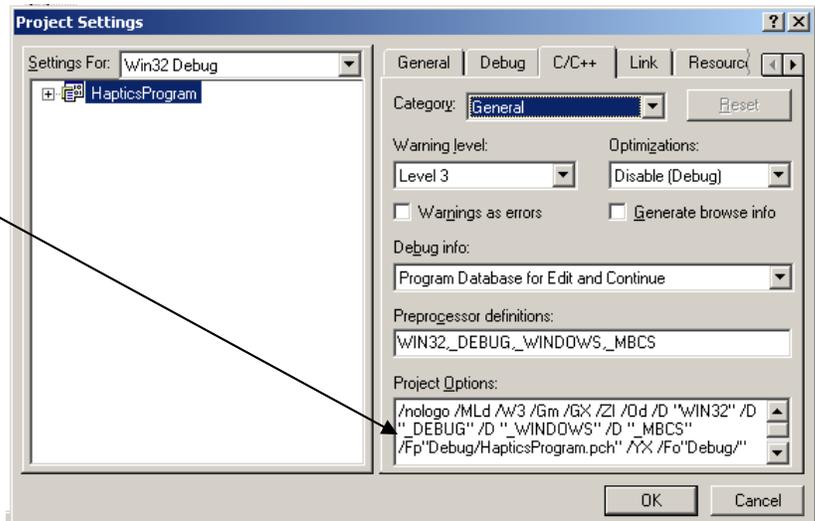
Click Project, Settings.

Click the "C/C++" tab.

Delete all text under "Project Options."

Replace with the following text:

```
/nologo /MDd /W3 /Gm /GX /ZI /Od /I "include"
/I "$(3DTOUCH_BASE)\include" /I
"$(3DTOUCH_BASE)\utilities\include" /D
"WIN32" /D "_DEBUG" /D "_CONSOLE" /D
"_MBCS" /Fo"Debug/" /Fd"Debug/" /FD /GZ /c
```



Click the "Link" tab.

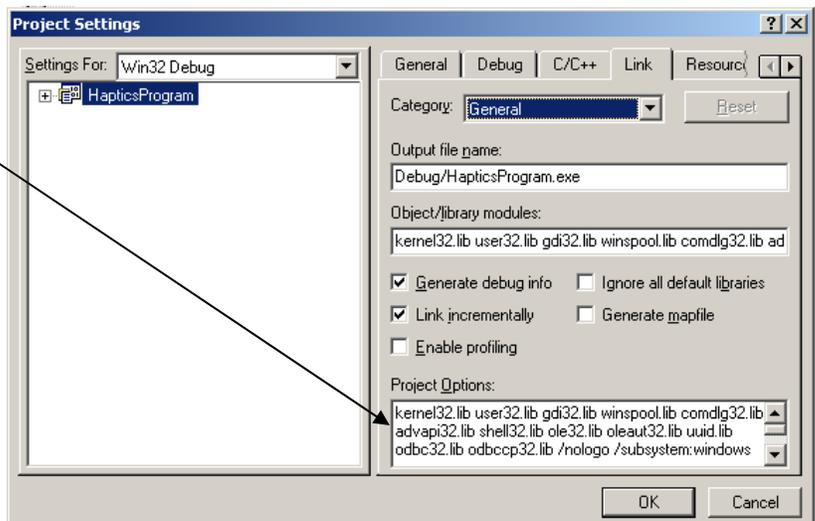
Delete all text under "Project Options."

Replace with the following text:

```
hl.lib hlud.lib hd.lib hdud.lib glut32.lib
opengl32.lib /nologo /subsystem:console
/incremental:yes /pdb:"Debug/Events.pdb"
/debug /machine:i386 /out:"Debug/Events.exe"
/pdbtype:sept /libpath:"$(3DTOUCH_BASE)\lib"
/libpath:"$(3DTOUCH_BASE)\utilities\lib"
```

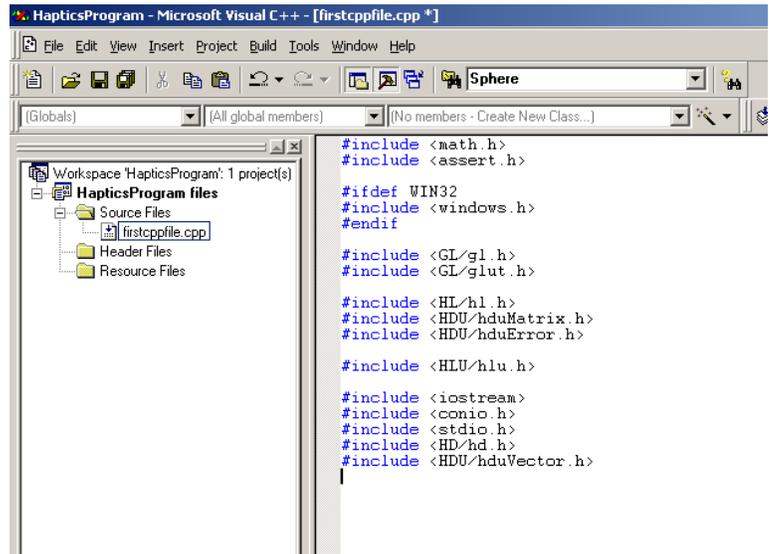
The "Object/library modules:" field will automatically populate with the libraries just added.

Click OK.



In your Source file, include the following headers:

```
#include <math.h>, #include <assert.h>, #ifdef WIN32, #include <windows.h>, #endif, #include <GL/gl.h>, #include <GL/glut.h>, #include <HL/hl.h>, #include <HDU/hduMatrix.h>, #include <HDU/hduError.h>, #include <HLU/hlu.h>, #include <iostream>, #include <conio.h>, #include <stdio.h>, #include <HD/hd.h>, #include <HDU/hduVector.h>
```



Add the following code to initialize the programs:

```
static HHD hHD = HD_INVALID_HANDLE;
static HHLRC hHLRC = 0;
```

```
HLuint Shapeld1;
```

```
#define CURSOR_SIZE_PIXELS 20
static double gCursorScale;
static GLuint gCursorDisplayList = 0;
```

```
void glutDisplay(void);
void glutReshape(int width, int height);
void glutIdle(void);
```

```
void exitHandler(void);
```

```
void initGL();
void initHL();
void initScene();
```

```
void drawSceneHaptics();
void drawSceneGraphics();
void drawCursor();
```

```
void updateWorkspace();
```

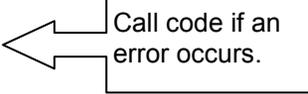
My understanding is that you need a new Shapeld for each object with a different force. Because we only have one object, only one Shapeld is needed.

Next, add Callback and Main Code:

```
HDCallbackCode HDCALLBACK FrictionlessSphereCallback(void *data)
{
    HDErrorInfo error;
    if (HD_DEVICE_ERROR(error = hdGetError()))
    {
        hduPrintError(stderr, &error, "Error during main scheduler callback\n");

        if (hdulsSchedulerError(&error))
        {
            return HD_CALLBACK_DONE;
        }
    }

    return HD_CALLBACK_CONTINUE;
}
```



Call code if an error occurs.

```
int main(int argc, char *argv[])
{
    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

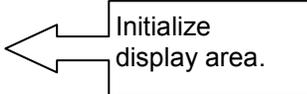
    glutInitWindowSize(500, 500);
    glutCreateWindow("First Haptics Program");

    glutDisplayFunc(glutDisplay);
    glutReshapeFunc(glutReshape);
    glutIdleFunc(glutIdle);

    atexit(exitHandler);

    initScene();
    glutMainLoop();

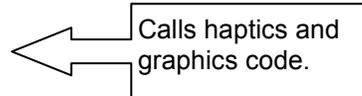
    return 0;
}
```



Initialize display area.

Add display codes:

```
void glutDisplay()
{
    drawSceneHaptics();
    drawSceneGraphics();
    glutSwapBuffers();
}
```



Calls haptics and graphics code.

```
void glutReshape(int width, int height)
{
    static const double kPI = 3.1415926535897932384626433832795;
    static const double kFovY = 40;

    double nearDist, farDist, aspect;
    glViewport(0, 0, width, height);

    nearDist = 1.0 / tan((kFovY / 2.0) * kPI / 180.0);
    farDist = nearDist + 2.0;
    aspect = (double) width / height;

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(kFovY, aspect, nearDist, farDist);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    gluLookAt(0, 0, nearDist + 1.0,
              0, 0, 0,
              0, 1, 0);

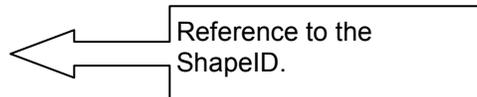
    updateWorkspace();
}
```

```
void glutIdle()
{
    glutPostRedisplay();
}
```

```
void exitHandler()
{
    hDeleteShapes(ShapeID1, 1);

    hMakeCurrent(NULL);
    if (hHLRC != NULL)
    {
        hDeleteContext(hHLRC);
    }

    if (hHD != HD_INVALID_HANDLE)
    {
        hdDisableDevice(hHD);
    }
}
```



Reference to the ShapeID.

```

void initScene()
{
    initHL();
}

void initGL()
{
    static const GLfloat light_model_ambient[] = {0.3f, 0.3f, 0.3f, 0.3f};
    static const GLfloat light0_diffuse[] = {0.7f, 0.7f, 0.7f, 0.7f};
    static const GLfloat light0_direction[] = {0.0f, 0.0f, 1.0f, 0.0f};

    glDepthFunc(GL_LEQUAL);
    glEnable(GL_DEPTH_TEST);

    glCullFace(GL_BACK);
    glEnable(GL_CULL_FACE);

    glEnable(GL_LIGHTING);
    glEnable(GL_NORMALIZE);
    glShadeModel(GL_SMOOTH);

    glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_FALSE);
    glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_FALSE);
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, light_model_ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_diffuse);
    glLightfv(GL_LIGHT0, GL_POSITION, light0_direction);
    glEnable(GL_LIGHT0);
}

void initHL()
{
    HDErrorInfo error;

    hHD = hdInitDevice(HD_DEFAULT_DEVICE);
    if (HD_DEVICE_ERROR(error = hdGetError()))
    {
        hduPrintError(stderr, &error, "Failed to initialize haptic device");
        fprintf(stderr, "Press any key to exit");
        getchar();
        exit(-1);
    }

    hHLRC = hICreateContext(hHD);
    hIMakeCurrent(hHLRC);

    hIEnable(HL_HAPTIC_CAMERA_VIEW);

    ShapeID1 = hIgenShapes(1);
        hITouchableFace(HL_FRONT);
}

```

Changing these parameters changes lighting on objects.

Reference to the ShapeID.

```

void updateWorkspace()
{
    GLdouble modelview[16];
    GLdouble projection[16];
    GLint viewport[4];

    glGetDoublev(GL_MODELVIEW_MATRIX, modelview);
    glGetDoublev(GL_PROJECTION_MATRIX, projection);
    glGetIntegerv(GL_VIEWPORT, viewport);

    hIMatrixMode(HL_TOUCHWORKSPACE);
    hILoadIdentity();

    /* fit haptic workspace to view volume */
    hIuFitWorkspace(projection);

    /* compute cursor scale */
    gCursorScale = hIuScreenToModelScale(modelview, projection, viewport);
    gCursorScale *= CURSOR_SIZE_PIXELS;
}

```

```

void drawSphere()
{
    glPushMatrix();
    glTranslatef(0,0,0);
    glutSolidSphere(0.5, 30, 30);
    glPopMatrix();
}

```

0,0,0 = position of object, I found that instead of x,y,z; it is actually x,z,y.
0.5 = radius, 30 = longitude lines, 30 = latitude lines.

Exercise: Change "glutSolidSphere(0.5,30,30)" to "glutSolidCube(0.5)." Your object will now be a solid cube when you run the program.

```

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

    initGL();
    drawCursor();
    drawSphere();
}

```

This graphics module calls the "drawSphere()" from above.

```

void drawSceneHaptics()
{
    hIBeginFrame();
    hIBeginShape(HL_SHAPE_FEEDBACK_BUFFER, ShapeID1);
    hITouchableFace(HL_FRONT);
    hIMaterialf(HL_FRONT_AND_BACK, HL_STIFFNESS, 0.3f);
    hIMaterialf(HL_FRONT_AND_BACK, HL_DAMPING, 0.1f);
    hIMaterialf(HL_FRONT_AND_BACK, HL_STATIC_FRICTION, 0.2f);
    hIMaterialf(HL_FRONT_AND_BACK, HL_DYNAMIC_FRICTION, 0.3f);

    drawSphere();

    hIEndShape();

    hIEndFrame();
}

```

Reference to the ShapeID.

Changes stiffness, damping, and friction factors.

This haptics module calls the "drawSphere()" from above as well.

```

void drawCursor()
{
    static const double kCursorRadius = 0.5;
    static const int kCursorTess = 15;
    HLdouble proxytransform[16];

    GLUquadricObj *qobj = 0;

    glPushAttrib(GL_CURRENT_BIT | GL_ENABLE_BIT | GL_LIGHTING_BIT);
    glPushMatrix();

    if (!gCursorDisplayList)
    {
        gCursorDisplayList = glGenLists(1);
        glNewList(gCursorDisplayList, GL_COMPILE);
        qobj = gluNewQuadric();

        gluSphere(qobj, kCursorRadius, kCursorTess, kCursorTess);

        gluDeleteQuadric(qobj);
        glEndList();
    }

    hlGetDoublev(HL_PROXY_TRANSFORM, proxytransform);
    glMultMatrixd(proxytransform);

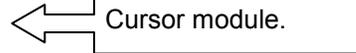
    glScaled(gCursorScale, gCursorScale, gCursorScale);

    glEnable(GL_NORMALIZE);
    glEnable(GL_COLOR_MATERIAL);
    glColor3f(0.0, 0.5, 1.0);

    glCallList(gCursorDisplayList);

    glPopMatrix();
    glPopAttrib();
}

```



Click Build, Compile.

If no errors, click Build, Run.

Appendix B

Notable Programming Code for Fig. 5 – Sphere behind Position Parameter Layer

```
HDCallbackCode HDCALLBACK FrictionlessSphereCallback(void *data)
```

```
{
const double sphereRadius = 20.0;
const hduVector3Dd spherePosition(40,0,-40);
const double sphereStiffness = 0.7;
const double sphereFriction = 0.2;
const double popthroughForceThreshold=5.0;
static int directionFlag = 1;
```

```
hdBeginFrame(hdGetCurrentDevice());
```

```
hduVector3Dd position;
hdGetDoublev(HD_CURRENT_POSITION, position);
```

```
int xvalue = position[0];
int yvalue = position[2];
int zvalue = position[1];
```

```
const double k = .4;
```

```
if (yvalue <= 0 && yvalue > -80)
```

```
{
double penetrationDistanceY = .2*fabs(position[2]);
hduVector3Dd forceDirectionY(0,0,1);
hduVector3Dd y = penetrationDistanceY*forceDirectionY;
hduVector3Dd fy = k*y;
hdSetDoublev(HD_CURRENT_FORCE, fy);
```

```
double distance = (position-spherePosition).magnitude();
```

```
if (distance < sphereRadius)
{
double penetrationDistance = sphereRadius-distance;
```

```
hduVector3Dd forceDirection = (position-spherePosition)/distance;
```

```
double k = sphereStiffness;
hduVector3Dd x = penetrationDistance*forceDirection;
hduVector3Dd f = k*x;
hduVector3Dd totalforce = f+fy;
hdSetDoublev(HD_CURRENT_FORCE, totalforce);
```

```

}
}
hdEndFrame(hdGetCurrentDevice());
return HD_CALLBACK_CONTINUE;
}
```

Note that position parameters are not intuitive. position (a,b,c) corresponds to (X axis, Z axis, Y axis)

penetrationDistance keeps track of how far the cursor moves in the y-direction.

Sets the direction of the force output. In this case, force will be towards +y direction.

Find the distance between the device and the center of the sphere.

If the user is within the sphere, i.e. if the distance from the user to the center of the sphere is less than the sphere radius, then the user is penetrating the sphere and a force should be commanded to repel him towards the surface.

Use $F=kx$ to create a force vector that is away from the center of the sphere and proportional to the penetration distance, and scaled by the object stiffness. Hooke's law explicitly.

Total force = force of the sphere (f) + force of the layer (fy)

Appendix B

Notable Programming Code for Figure 7 – Final Graphical Design

```
void drawSphere()
{
    glPushMatrix();
    glTranslatef(-.3,0,-.3);

    glutSolidSphere(0.30, 30, 30);
    glPopMatrix();
}
```

Characteristics for hidden sphere.

```
void drawSphere2()
{
    static const GLfloat light_model_ambient2[] = {0.3f, 0.3f, 0.3f, 1.0f};
    static const GLfloat light0_diffuse2[] = {0.1f, 0.9f, 0.9f, 0.0f};
    static const GLfloat light0_direction2[] = {0.0f, -0.4f, 1.0f, 0.0f};

    glPushMatrix();
    glTranslatef(1.25,-1,-.2);
    glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_FALSE);
    glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_FALSE);
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, light_model_ambient2);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_diffuse2);
    glLightfv(GL_LIGHT0, GL_POSITION, light0_direction2);
    glEnable(GL_LIGHT0);
}
```

Color characteristics for sphere on bottom right.

```
glutSolidSphere(0.18, 30, 30);
glPopMatrix();
}
```

```
void drawCube()
{
    glutSolidCube(1.0);
    glPopMatrix();
}
```

Cube 1 – Far left

```
void drawCube2()
{
    glPushMatrix();
    glTranslatef(0.0,0,-.6);
    glutSolidCube(1.0);
    glPopMatrix();
}
```

Cube 2 – Middle

```
void drawCube3()
{
    glPushMatrix();
    initGL();
    glTranslatef(1,0,-.6);
    glutSolidCube(1.0);
    glPopMatrix();
}
```

Cube 3 – Far right

```

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

drawCursor();
//drawSphere();
drawSphere2();
drawCube();
drawCube2();
drawCube3();
}

```

drawSphere() is commented out so that graphics are not drawn for it.

```

void drawSceneHaptics()
{
hlBeginFrame();
hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, sphereShapeId);
hlTouchableFace(HL_FRONT);
hlMaterialf(HL_FRONT_AND_BACK, HL_STIFFNESS, 0.4f);
hlMaterialf(HL_FRONT_AND_BACK, HL_STATIC_FRICTION, 0.2f);
hlMaterialf(HL_FRONT_AND_BACK, HL_DYNAMIC_FRICTION, 0.3f);
drawSphere();
hlEndShape();

```

number determine's stiffness of objects.

```

hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, sphereShapeId2);
hlTouchableFace(HL_FRONT);
hlMaterialf(HL_FRONT_AND_BACK, HL_STIFFNESS, 0.4f);
hlMaterialf(HL_FRONT_AND_BACK, HL_STATIC_FRICTION, 0.2f);
hlMaterialf(HL_FRONT_AND_BACK, HL_DYNAMIC_FRICTION, 0.3f);
drawSphere2();
hlEndShape();

```

```

hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, cubeShapeId);
hlTouchableFace(HL_FRONT);
hlMaterialf(HL_FRONT_AND_BACK, HL_STIFFNESS, 0.25f);
hlMaterialf(HL_FRONT_AND_BACK, HL_STATIC_FRICTION, 0.2f);
hlMaterialf(HL_FRONT_AND_BACK, HL_DYNAMIC_FRICTION, 0.3f);
drawCube();
hlEndShape();

```

Stiffness of first two cubes are the same.

```

hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, cube2ShapeId);
hlTouchableFace(HL_FRONT);
hlMaterialf(HL_FRONT_AND_BACK, HL_STIFFNESS, 0.25f);
hlMaterialf(HL_FRONT_AND_BACK, HL_STATIC_FRICTION, 0.2f);
hlMaterialf(HL_FRONT_AND_BACK, HL_DYNAMIC_FRICTION, 0.3f);
drawCube2();
hlEndShape();

```

```

hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, cube3ShapeId);
hlTouchableFace(HL_FRONT);
hlMaterialf(HL_FRONT_AND_BACK, HL_STIFFNESS, 0.7f);
hlMaterialf(HL_FRONT_AND_BACK, HL_STATIC_FRICTION, 0.8f);
hlMaterialf(HL_FRONT_AND_BACK, HL_DYNAMIC_FRICTION, 0.1f);
drawCube3();
hlEndShape();

```

Stiffness and friction of third cube is highest.

```

hlEndFrame();

```

```
/*void drawSceneHaptics0()
{
    hlBeginFrame();
    hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, sphereShapeId);
    hlTouchableFace(HL_FRONT);
    drawSphere();
    hlEndShape();
    hlEndFrame();
}

void drawSceneHaptics2()
{
    hlBeginFrame();
    hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, cubeShapeId);
    hlTouchableFace(HL_FRONT);
    drawCube();
    hlEndShape();
    hlEndFrame();
}

void drawSceneHaptics3()
{
    hlBeginFrame();
    hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, cube2ShapeId);
    hlTouchableFace(HL_FRONT);
    drawCube2();
    hlEndShape();
    hlEndFrame();
}
```

Appendix C

Notable Code for Automated Graphics Design

```
double cubesize = 0.2;
int    numberofrows = 6;
int    numberofcols=6;
```

cubesize = height,width,depth of cube
numberofrows = height of final stack of cubes
numberofcols=width of final stack of cubes

```
double drawCube(double j, double k)
{
  initGL();
  glPushMatrix();
  glTranslatef(-1+j,-0.5+k,-.5);
  glutSolidCube(cubesize);
  glPopMatrix();
  return j,k;
}
```

Code of cube that will be multiplied.
“glTranslatef” shows initial position.

```
void drawSceneGraphics()
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  drawCursor();
```

```
for(double i=0; i<(numberofcols*cubesize);i=i+cubesize)
  {
    for(double h = 0; h<=(numberofrows*cubesize); h=h+cubesize)
      drawCube(i,h);
  }
```

Graphics and Haptics
portions are similar.
Both builds a plane
“numberofrows” high
by “numberofcols”
wide.

```
void drawSceneHaptics()
{
  hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, Shaped1);
  hlTouchableFace(HL_FRONT);
  hlMaterialf(HL_FRONT_AND_BACK, HL_STIFFNESS, 0.7f);
  hlMaterialf(HL_FRONT_AND_BACK, HL_DAMPING, 0.1f);
  hlMaterialf(HL_FRONT_AND_BACK, HL_STATIC_FRICTION, 0.2f);
  hlMaterialf(HL_FRONT_AND_BACK, HL_DYNAMIC_FRICTION, 0.3f);

  for (double m=0; m<(numberofcols*cubesize);m=m+cubesize)
  {
    for (double n=0; n<=(numberofrows*cubesize); n=n+cubesize)
      drawCube(m,n);
  }

  hlEndShape();
```