HAPTIC ENABLED MULTIDIMENSIONAL CANVAS

A Thesis

Presented to

The Graduate Faculty of The University of Akron

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Kalyan Thokala

July, 2011
HAPTIC ENABLED MULTIDIMENSIONAL CANVAS

Kalyan Thokala

Thesis

Approved:  __________________________________________

Advisor
Dr. Yingcai Xiao

Accepted:  __________________________________________

Dean of Arts and Sciences
Dr. Chand Midha

Committee Member
Dr. Eunsu Kang

Dean of the Graduate School
Dr. George Newkome

Committee Member
Mr. Matthew Kolodziej

Date

Department Chair
Dr. Chien-Chung Chan
ABSTRACT

An artist needs a canvas to paint or draw to convey a message in the form of variable color composition in an aesthetic sense. A traditional canvas allows the artist to produce and preserve her work, which exists in two dimensions. This project is about building a software tool which allows an artist to explore further dimensionality of canvas through haptic perception. This haptic enabled tool supports multi-dimensional canvas for artist to work not only in 2D but also in 2.5D and 3D. For each dimension, an artist uses a canvas of different haptic feeling and each dimensionality allows added flexibilities in creating artworks.
ACKNOWLEDGEMENTS

I am very thankful to my parents, for encouraging and supporting me in pursuing my higher studies and also making this thesis possible.

I would like to acknowledge the professors who inspired me throughout my masters program. Dr. Eunsu Kang and Mr. Matthew Kolodziej, thank you very much for inspiring me through arts and supporting me in developing this software tool. I am very thankful to Dr. Chien-Chung Chan for his advice throughout my entire master program. Dr. Yingcai Xiao, thanks a lot for being my professor and advisor, giving the opportunity at the same time motivating, encouraging and supporting me in the entire program.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS</td>
<td>xi</td>
</tr>
<tr>
<td><strong>CHAPTER</strong></td>
<td></td>
</tr>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Survey of Previous Work</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Outline of Thesis</td>
<td>3</td>
</tr>
<tr>
<td>II. BACKGROUND</td>
<td>5</td>
</tr>
<tr>
<td>2.1 New Media Art</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Overview of Haptic Devices</td>
<td>5</td>
</tr>
<tr>
<td>2.3 PHANTOM Omni Device</td>
<td>7</td>
</tr>
<tr>
<td>2.4 Overview of OpenGL</td>
<td>10</td>
</tr>
<tr>
<td>2.5 OpenGL Program Logic Cycle</td>
<td>10</td>
</tr>
<tr>
<td>III. DESIGN AND IMPLEMENTATION</td>
<td>12</td>
</tr>
<tr>
<td>3.1 Haptic Canvas Application Architecture</td>
<td>12</td>
</tr>
<tr>
<td>3.2 HCA Resource Sharing and Event Processing</td>
<td>13</td>
</tr>
<tr>
<td>3.3 OpenHaptics Program Logic Cycle</td>
<td>14</td>
</tr>
<tr>
<td>3.4 Steps Involved in Haptic Canvas Drawing</td>
<td>15</td>
</tr>
<tr>
<td>3.5 Design and Implementation of 2D Haptic Canvas</td>
<td>18</td>
</tr>
<tr>
<td>3.5.1 Type Defined Structures</td>
<td>19</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>1. Sensable® Haptic Device</td>
<td>7</td>
</tr>
<tr>
<td>2. Phantom’s Six Degrees of Freedom</td>
<td>8</td>
</tr>
<tr>
<td>3. Inputs/Outputs of Haptic Device</td>
<td>8</td>
</tr>
<tr>
<td>4. Movements of Haptic Device</td>
<td>9</td>
</tr>
<tr>
<td>5. Sample OpenGL Program</td>
<td>10</td>
</tr>
<tr>
<td>6. HCA Architecture</td>
<td>12</td>
</tr>
<tr>
<td>7. Event Processing in OpenHaptics Environment</td>
<td>13</td>
</tr>
<tr>
<td>8. OpenHaptics Program Logic Cycle</td>
<td>14</td>
</tr>
<tr>
<td>9. HCA Drawing Sequence</td>
<td>15</td>
</tr>
<tr>
<td>10. Type Defined Structures of 2D Canvas</td>
<td>19</td>
</tr>
<tr>
<td>11. Class Diagrams of 2D Canvas</td>
<td>23</td>
</tr>
<tr>
<td>12. State Diagram Showing Haptic Events in 2D Canvas</td>
<td>24</td>
</tr>
<tr>
<td>13. Graphic Scene from 2.5D Canvas System</td>
<td>27</td>
</tr>
<tr>
<td>14. Type Defined Structures of 2.5D Canvas</td>
<td>28</td>
</tr>
<tr>
<td>15. Class Diagram of 2.5D Canvas</td>
<td>30</td>
</tr>
<tr>
<td>16. State Diagram of Haptic Events in Canvas Modifying Mode</td>
<td>31</td>
</tr>
<tr>
<td>17. Image Sequence Showing Canvas Deformation</td>
<td>32</td>
</tr>
<tr>
<td>18. Mesh Diagram Showing Canvas Coordinates</td>
<td>33</td>
</tr>
<tr>
<td>19. Mesh Diagram Showing Centered Collided Vertex and Surrounding Vertexes</td>
<td>33</td>
</tr>
</tbody>
</table>
20. State Diagram of Haptic Events in Painting Mode ........................................... 34
21. Ring Structure and the Orientation of its Plane to Stylus Proxy ...................... 35
22. Tracking Vertex Planes According to the Stylus Movements ...................... 36
23. Group of Planes ............................................................................................. 36
24. Process of Generating Quad Strip Using OpenGL API .................................. 37
25. Cylindrical Structure for Wet Medium .......................................................... 38
26. Cylindrical Structures after Surface Generation .......................................... 38
27. Illustration of Free Fall with Re-Orientiation of Vertex Plane ......................... 39
28. Free Fall Simulation with Re-Orientiation of Vertex Plane ............................ 39
29. The 3D Canvas ................................................................................................ 40
30. Type Defined Structures in 3D Haptic Canvas .............................................. 41
31. State Diagram of Haptic Events in Creating Canvas ...................................... 42
32. Axis of Rotation ............................................................................................... 43
33. Circle Symmetrical 3D Canvas ....................................................................... 43
34. Circle Asymmetrical 3D Canvas .................................................................... 44
35. Illustration of Vertex Tracing .......................................................................... 45
36. System Setup ................................................................................................... 47
37. Device Test Utility Program ........................................................................... 48
38. Device Auto Calibration .................................................................................. 48
39. 2D Haptic Canvas GUI ................................................................................... 49
40. Software Testing by Kolodziej ........................................................................ 49
41. Monumental Stair Case by Kolodziej ............................................................. 50
42. Individual Components of the Artwork ......................................................... 50
43. 2.5D Haptic Canvas GUI ............................................................................... 52
44. 2.5D Canvas Deformation .............................................................................. 52
45. 2.5D Canvas Painting ...................................................................................... 53
46. 3D Haptic Canvas GUI ................................................................................... 54
<table>
<thead>
<tr>
<th></th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td>Monumental Stair Case in 3D Canvas</td>
<td>55</td>
</tr>
<tr>
<td>48</td>
<td>Monumental Stair Case in 3D Canvas Showing Regions of Transparency</td>
<td>56</td>
</tr>
<tr>
<td>49</td>
<td>Complex Structures I Rendered by Kang</td>
<td>56</td>
</tr>
<tr>
<td>50</td>
<td>Complex Structures II Rendered by Kang</td>
<td>57</td>
</tr>
<tr>
<td>51</td>
<td>Simple Structures Rendered by Kang</td>
<td>57</td>
</tr>
<tr>
<td>52</td>
<td>Surface Painting</td>
<td>58</td>
</tr>
<tr>
<td>53</td>
<td>Illustration of Portrait at Different Angles in 3D Space</td>
<td>58</td>
</tr>
<tr>
<td>54</td>
<td>Sample Art Data File</td>
<td>59</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
<td></td>
</tr>
<tr>
<td>HDAPI</td>
<td>Haptic Device API</td>
<td></td>
</tr>
<tr>
<td>HLAPI</td>
<td>Haptic Library API</td>
<td></td>
</tr>
<tr>
<td>PDD</td>
<td>PHANTOM Device Drivers</td>
<td></td>
</tr>
<tr>
<td>HCA</td>
<td>Haptic Canvas Application</td>
<td></td>
</tr>
<tr>
<td>HDU</td>
<td>Haptic Device Utility</td>
<td></td>
</tr>
<tr>
<td>HLU</td>
<td>Haptic Library Utility</td>
<td></td>
</tr>
<tr>
<td>HHLRC</td>
<td>Haptic Library Haptic Rendering Context</td>
<td></td>
</tr>
<tr>
<td>HHD</td>
<td>Haptic Device Handle</td>
<td></td>
</tr>
<tr>
<td>HIP</td>
<td>Haptic Interface Point</td>
<td></td>
</tr>
<tr>
<td>DOF</td>
<td>Degree Of Freedom</td>
<td></td>
</tr>
<tr>
<td>OpenGL</td>
<td>Open Graphics Library</td>
<td></td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
<td></td>
</tr>
<tr>
<td>GLUT</td>
<td>OpenGL Utility Toolkit</td>
<td></td>
</tr>
<tr>
<td>3D</td>
<td>Three Dimension</td>
<td></td>
</tr>
<tr>
<td>2.5D</td>
<td>Two and Half Dimension</td>
<td></td>
</tr>
<tr>
<td>2D</td>
<td>Two Dimension</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

1.1 Motivation

Painting is a process of conveying an artist’s imagination and interpretation of a subject through applying a visible pigment onto a canvas in an aesthetic way, which has the quality to describe the expressive features in terms of visual elements of arts [32]. The visible pigments in the painting are made up of different materials, which are available in different media to give the appropriate color composition in the art. The wet medium of materials produces a very little contact friction on the surface of a canvas. On the other hand dry medium of materials produces a good contact friction on canvas. The contact friction between the canvas and the material medium is determined by the adhesive properties of material medium and the roughness of canvas. The canvas has a property by which it holds the pigments of paint together preserving the artwork.

The conventional canvas exists in two dimensional forms having some material properties and allows the artists in mastering 2D paintings. In digital world, these canvases are rendered using computer graphics and the process of painting is achieved through modern technological tools. This approach introduces the concept of virtual arts, which are a substitute for valuable natural resources. The advent of haptic technology
introduced the concept of sense of touch in virtual arts. The arts created with modern technologies including haptic perception is a mere replacement of traditional arts. The project Haptic Enabled Multidimensional Canvas is about developing a software tool which helps in exploring different dimensionality of canvas and mastering painting process on such dimensionalities. Through this project the artist can paint virtually onto a canvas having different physical properties with haptic perception. The device used in this project provides the force resistance applied on to any virtual object at any given point of time. This mechanism allows gathering and visualizing the pressure points applied by the artist on the canvas.

1.2 Survey of Previous Work

The advent of haptic devices has triggered conducting research in many areas. One of the research areas includes Fine Arts and Sculpture. The device usefulness is found in creating digital arts with the help of tactile feedback interaction providing the artist a virtual environment in which the artwork can be felt while manipulating the device in 3D space. From the beginning, research was conducted in building the software tools which produce a realistic brush patterns to simulate the natural brush strokes applied on a canvas. Artist can experience the feel of virtual brushes while painting on the canvas surface {1 Baxter, W.V. 2004};{8 Jeng-sheng Yeh 2002}. These software tools simulate the natural way of producing brush patterns over 2D canvas. The haptic devices can also be used to develop a software tool which can interact with 3D mesh models. Gregory developed a tool with which an artist can edit and paint on a 3D polygonal mesh model {6 Gregory A.D. 2000}. Yongxiao developed a local dynamic
subdivision algorithm to manipulate the surface of 3D mesh models in order to create a
paint process {18 Yongxiao Fu. 2008}. Johnson developed a tool which maps the 2D
texture onto a 3D model, they used an adaptive brush size mechanism which aligns the
size of brush according to the shape of the 3D model and by this process the 2D textures
are mapped uniformly onto 3D models without any deformation {9 Johnson,D. 1999}. 
Foskey developed a two handed Visio-Haptic mixed reality systems, with the help of the
haptic device to interact with 3D virtual objects to manipulate and paint on its surfaces
{{5 Foskey,M. 2002}; {3 Faeth,A. 2008}]. Laehyun developed a tool which can sense
the dynamic materialistic properties of both paint and 3D objects with the help of the
haptic device {{11 Laehyun Kim 2003}]. Haptic devices are also used to develop a
virtual sculpting of 3D objects {{4 Fengxin Yan 2009}; {19 Xiaobo P. 2009}]. Sulaiman
presented a study conducted on degree of likeness by the artist’s interaction with virtual
objects having different materialistic properties {17 Sulaiman, Suziah 2010}.

1.3 Outline of Thesis

This report consists of detailed work explained through various chapters. Chapter
I includes the information regarding the motivation based on survey of previous work.
Chapter II includes the background study on the technology with which the project is
done. Chapter III discusses about the design and implementation of this software tool.
Each design regarding the multidimensional canvases is explained in detail with the help
of state diagrams and methods. Chapter IV discusses about the case studies in which a
step by step procedure are explained pictorially with the help of screen shots. Each
dimensionality is handled in different case studies, these are the artworks produced by
different artists. Chapter V discusses the summary of the work and how the system can be utilized, the modifications and future work is also explained. Appendix and bibliography are included at the end.
CHAPTER II

BACKGROUND

2.1 New Media Art

Arts evolved through human history taking different levels of complexity with different media. Traditional painting requires pigments, brushes and canvas through which artists can create their works. In the digital age, the artist can produce the similar work virtually using modern digital tools. They may even produce more interesting patterns based on the flexibility of materials and tools in the virtual environment. These new media including digital technologies extend the definition of traditional arts. The New Media Art uses the processing power of computer to produce digital artworks {27, 28}.

New media Art requires a lot of interaction between the artwork and participant. The haptic devices are one of the modern technological inventions which are helpful in creating New Media Art. William developed a software tool which simulates the functionality of Theremin using haptic device, the music produced in the system is controlled by the movements of device stylus in 3D space {20}. 
2.2 Overview of Haptic Devices

Haptic is related to the sense of touch and these devices are a revolutionary in tactile feedback technology. The general concept of these devices is to give a feel of the virtual object in computer graphics and to understand the physical attributes in the virtual world. The presences of these virtual objects are detected and fed to the user with a mechanical resistance generated in the device. The perception of the virtual object in feedback type varies upon the applications which are developed with it \{21\}. Based on the inputs and outputs, the haptic device takes many forms. This device has physical limits, upon which user can detect no resistance or sudden obstruction of movement which causes the device to stop the flow of action in certain directions. This limit in space called a haptic room or haptic space where a user can define virtual shapes which are defined graphically onto the computer screen.

The haptic technology is a new branch in modern technology where extensive research is going on to make the device more sophisticate and more applicable. The following list shows some of the industrial applications using this technology.

- Assembly Path Planning.
- Automotive.
- Ceramics.
- Dental.
- Film and Media.
- Fine Arts and Sculpture.
- Graphic Designing and Modeling.
• Medical.
• Molecular Modeling.
• Research.
• Robotics.
• Simulation & Visualization.
• Training.

2.3 PHANTOM Omni Device

Figure 1 shows the Phantom Omni® device from Sensable® technologies. This device comes with six degrees of freedom in space.
Figure 2: Phantom’s Six Degrees of Freedom

Figure 2 shows the possible movements of the mechanical arm in different directions in 3D space. The first three movements represent the device stylus position in 3D space and the last three represent the orientation with ‘x’, ‘y’ and ‘z’ axis respectively. Each degree of freedom has a limit on the rotational parameters which defines a limited space with which a device can operate safely and properly.

Figure 3: Inputs/Outputs of Haptic Device

Figure 3 shows the inputs and outputs of the device, the PHANTOM Omni gives 6 input readings according to the device status. The input readings are the position and orientation of device stylus at any point of time. The device gives the output in terms of
force resistance in particular direction and sets the device stylus in “x”, “y”, and “z” location with some inertial force.

Figure 4: Movements of Haptic Device

Figure 4 shows the device macro and micro movements, the macro movements gives the device stylus in particular “x”, “y” and “z” location. The micro movements give the orientation of stylus in “x”, “y” and “z” axis respectively.

The Phantom Omni device is supplied with the OpenHaptics toolkit, which has important Application Programming Interfaces (API’s) for developers. The API’s include Haptic Device API (HDAPI) and Haptic Library API (HLAPI). The HDAPI provides low-level access to the haptic device, enabling the haptics programmer to render forces directly. It offers programmers to control the configuration of runtime behavior of the drivers, and also provides a convenient utility features and debugging aids. The HLAPI provides high-level haptic rendering and is designed to be familiar to OpenGL® API
programmers. It allows significant reuse of existing OpenGL code and greatly simplifies synchronization of the haptics and graphics threads [21].

2.4 Overview of OpenGL

OpenGL stands for Open Graphics Library, a standard graphic application programming interface which is being managed by Khronos group. OpenGL is designed to write applications which work on different platforms. Since its introduction in 1992 it is widely used in standalone and interactive 2D and 3D graphics applications. OpenGL includes a broad set of features including rendering, texture mapping, special effects and other visualization techniques. OpenGL simplifies software development by providing geometric and image primitives, display lists, model transformations, lighting, texturing, shading, blending and other features [22].

OpenGL supports all operating systems including UNIX, Windows and MacOS. It can be called from C, C++, FORTRAN, Python, Perl and Java.

2.5 OpenGL Program Logic cycle

The main program logic cycle looks like in the Figure 5.

```c
void main(int argc, char **argv) {
    glutInit(&argc, &argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(320, 320);
    glutCreateWindow("OpenGL");
    glutDisplayFunc(renderScene);
    glutMenuFunc(renderScene);
    glut3dFunc(renderScene);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```

Figure 5: Sample OpenGL Program.
In OpenGL programming each API starts with a glut word. The function “glutInit” is used to initialize the environment variables passed from operating system and function “glutInitDisplayMode” is used to setup the display mode by allocating memory for the respective display buffer passed as parameters. The function “glutWindowPosition” is used to specify the initial location of window and function “glutInitWindowSize” specifies the window size in length and height in unit pixels. The function “glutCreateWindow” is used to specify title for a window and create a window. The function “glutDisplayFunc” is used for what to display, all geometry are specified in this routine. The function “glutIdleFunc” is useful in creating animations, calling an idle function between the loops creates the animation effect. The function “glutReshapeFunc” is called whenever the size and position of the display window is changed. In this function all operations are done to restore the original perspective. The function “glutMainLoop” is called to listen for events occurred during course of runtime of program.
CHAPTER III

DESIGN AND IMPLEMENTATION

3.1 Haptic Canvas Application Architecture

Haptic Canvas Application (HCA) Architecture follows the OpenHaptics Application Framework. The toolkit comes with API’s such as Haptic Library API (HLAPI) and Haptic Device API (HDAPI). HLAPI is built on top of HDAPI and is used to specify the geometry of graphical objects in haptic rendering, whereas HDAPI is used to interact directly with device modifying its rate of servo loop. A servo loop is maintained in a high priority thread in order to compute the forces for the device. This loop executes at a high update rate of 1 KHz to maintain the forces rendered in the device.

![Figure 6: HCA Architecture](image_url)
Figure 6 gives an overview of OpenHaptics application framework. The user application is built using OpenGL APIs and OpenHaptics API’s. The geometry which is specified in OpenGL context is used with HLAPI’s to render objects haptically. The haptic device must be initialized through HDAPI before use. The Haptic Device Handle (HHD) is used from HDAPI by the Haptic Library Haptic Rendering Context (HHLRC) to establish an interface with the device. A servo loop is established between device and the application. This is run by Phantom Device Drivers (PDD). The graphic loop which runs at a lower rate than servo loop is established between the application and graphic display unit.

3.2 HCA Resource Sharing and Event Processing

![Diagram of Event Processing in OpenHaptics Environment](image)

Figure 7: Event Processing in OpenHaptics Environment

Figure 7 shows how events are processed in the application. The events which are generated for graphics and haptics are handled separately with respective callback
function. This callback function may utilize the global data as a message passing variables between the callback functions.

### 3.3 OpenHaptics Program Logic Cycle

This application is developed using OpenGL and OpenHaptics API’s. OpenGL is a standard graphics interface between operating system and graphics hardware. It is platform independent, once written on one system, it can be executed anywhere with different configurations. OpenGL commands are used to build heavy graphical applications, which are completely dependent on basic graphics primitives such as vertex, lines, polygons and etc. OpenHaptics is a software development toolkit for building applications and interacting with the haptics device. OpenHaptics is provided by Sensable Technologies.

![Diagram of OpenHaptics Program Logic Cycle](#)

**Figure 8: OpenHaptics Program Logic Cycle**
Figure 8 shows the program logic cycle. All OpenGL parameters including lighting, shading, and material properties are initialized in first step. The haptic device is initialized in a separate routine, with specifying what events have to be handled by giving their callback function signatures. Also the object id’s which are going to be created in application are specified here. A routine for rendering the graphic object onto screen with the help of OpenGL display lists follows. Then a routine for specifying materials properties of graphic objects and their geometry is specified in haptic rendering. The events are handled once they are generated in application and processed with respective callback functions. A procedure for updating the geometry of graphical objects, then a procedure for updating force values in the device occurs. This runs in a loop until the application terminates.

### 3.4 Steps Involved in Haptic Canvas Drawing

Figure 9: HCA Drawing Sequence.

The following are the steps involved in drawing using haptic device which are showed in Figure 9.

- Capturing and Storing Device Attributes
- Rendering Graphic Scene
- Rendering Haptic Scene
- Event Processing
Capturing and Storing Device Attributes

The haptic device is provided with the haptic toolkit API’s through which device attributes are read time to time when in use. One such API is hlCacheGetDoublev, it is an HLAPI built in function through which the dynamic attributes of a proxy are cached. There are different enumerators such as HL_PROXY_POSITION, HL_PROXY_ROTATION and HL_DEVICE_FORCE which are passed as a parameter to identify what device attributes are to be cached. The HL_PROXY_POSITION is used to capture the current location of haptic stylus in 3D space, the HL_PROXY_ROTATION is used to record the orientation of the stylus with respect to 3D coordinates and HL_DEVICE_FORCE is generally used to read the force resistance generated in a device when the stylus hits any haptic object in space. All these device attributes are possible to capture anywhere in the haptic event loop. There is a specific event id called HL_EVENT_MOTION which is triggered when ever the device stylus is in motion. The best possible way to record device attributes is to call the API’s with respective parameters with in an HL_EVENT_MOTION event callback to get the accurate device location.

The capture of device attributes are based on a flag which is set on/off based on a event id called HL_EVENT_TOUCH, this event is triggered when ever a proxy touches any object in haptic space. The virtual canvas which is rendered in haptics space is defined with specific material properties so as to give different tactile resistance. All device attributes are read and copied to the members of a new linked list structure node which is specifically used to store the device attributes. This linked list is defined globally so as to access any where in the program.
**Rendering Graphic Scene**

Rendering graphic scene is to mention all the positions and orientations of each graphical display entities. In this painting application the most important things to display are canvas, the drawing pencil, and the sketch. OpenGL has a specific way to clear the buffer bit before displaying anything on to the screen. This is done by calling an OpenGL API “glClear” by passing parameter as GL_COLOR_BUFFER_BIT. With this API the screen is updated with new information by erasing color and depth information, this is very important while animating any graphical entities on the screen. In animation the screen has to be cleared before printing the graphic entities with new coordinate locations. To get a smooth animation effect the scene has to be refreshed with new coordinate locations of graphic entities at a rate of 24 frames per second. All graphic entities which are stored as display lists so as to access them very fast while displaying on to the screen. The sketch which is being drawn on to the screen is displayed from two stored procedures, one from the display lists and the other from the linked lists.

**Rendering Haptic Scene**

The haptic scene creates a virtual environment with all graphic entities so as to get the feel of object presence in the device space when interacting with the stylus. The haptic scene is called after graphic scene in an application loop so as to sync with graphic rendered objects where the collisions are detected accurately.

**Event Processing**

The haptic, OpenGL, windows, keyboard and mouse events are processed after respective events generated in the application. All event processing call back functions
communicate with other functions in the application using global access variables. These variables store the status of application.

3.5 Design and Implementation of 2D Haptic Canvas

A 2D virtual canvas is rendered as a real canvas having a fixed “z” value. The canvas has open surface area where an artist can drag the device stylus. All strokes related to the drawing are recorded on this surface area. In general to draw something on to canvas requires a movement of brush or a pen. The haptics stylus position is used as a proxy for brush animation in this graphics application. The medium in this tool is just an imitation of charcoal and pastel brush, which records fine shades of grey and color values upon the tactile feedback from the device stylus.

Charcoal and pastels are used as a dry media in arts. These are available in various hardness, which determines the fine shades and quality while drawing. The charcoal powder is mixed with some adhesives to transform into a hard material which can then be made into pencils or sticks of various shapes and sizes. The amount of adhesive materials determines the hardness of media \{31\}. Canvas is made of different materials generally to hold the media particles to represent a stroke of an artist. The surface of canvas is made so as to provide friction and holding capacity of material. In virtual canvas the object is drawn as cube which is scaled in order to get a look of 2D canvas. The tactile feel of canvas is produced when a stylus proxy touches the surface of the canvas.
3.5.1 Type Defined Structures

The following type defined structures shown in Figure 10 gives the information related to the member variables and their types. These type defined structures are used to create dynamic linked lists to record and maintain the data throughout the program.

![Type Defined Structures of 2D Canvas](image)

Each artist stroke at a moment is captured and saved in a member variable of a type defined structure which is specially defined for creating linked lists. This “artistStroke_node” structure contains all necessary information of device attributes at a particular location for further processing. The “x”, “y” and “z” members of structure store the location of the current stroke are stored in double type in order to capture location more precisely, the “brush_gAngle_x”, “brush_gAngle_y” and
“brush_gAngle_z” stores the orientation of the haptic stylus in 3D space. The member “brushForce” records the current force resistance developed in a device when the stylus proxy touches any object. In this case the force resistance is recorded at particular point when stylus proxy touches the canvas while drawing. The member variables “brushVelocity_x”, “brushVelocity_y” and “brushVelocity_z” stores the current velocity of stylus at that location and recording the speed of particular stroke. The member variables “canvasTouchLocation_x”, “canvasTouchLocation_y” and “canvasTouchLocation_z” stores the point in 3D location when ever the stroke begins on canvas by touching it, each artist stroke as starting point and ending point, the starting point denotes where artist touches the canvas and begins drawing. This stroke may be a continuous movement of stylus without untouching the canvas. During the course of the stroke moment, all individual points in the stroke are captured and recorded simultaneously. The member variable “redComponent”, “greenComponent” and “blueComponent” are the values of RGB color components which represents the pigment of color which is applied on to the canvas, these component values are modified through the key board interface at any point of time during program execution giving the oppurtunity to illustrate the use of various shades of color on canvas. The variable “colorIntensity” is a calculated based on the amount of force applied. It ranges from 0.0 to 1.0. For lighter force applied the “colorIntensity” value is set to near 0.0 representing the translusent properties. For heavier force the value is set to near 1.0 representing the opaqueness properties. The intensity of the color component is directly proportional to the force applied. The member variable “zDifference” stores the difference of the “z” coordinate calculated from the canvas location and stroke touching point. This variable
gives the penetration depth on the canvas. The “brushMode” variable stores the type of media in a particular stroke. In this application charcoal pencil and pastel are used as a medium of drawing. The charcoal requires the same values in all three components of color to represent the grey values and pastel requires color information to highlight the particular color shade. The “pointSize” member variable stores the information of the radius of the base area of the stroke. This information is required to demonstrate the spreading nature of medium on canvas. The member variable “strokeCount” stores count information, this value records the current stroke number on the canvas. The member “starting_node_of_artistCharcoalParticals” stores the pointer to a location of a list where all random color pigments are stored for a particular location. This represents the cloud of particles around the location to represent the color pigment coverage area. The number of nodes in this list represents the thickness of medium applied to the canvas.

Figure 10 shows the “record_artistStroke_node” structure for tracing the stylus proxy throughout the program and stores the location of medium applied on canvas by coordinates and other attributes. All members in this structure are of the same purpose as mentioned in structure “artistStroke_node”, except “starting_node_of_artistCharcoalParticals” this is ignored for tracing the stylus on canvas.

The structure “artistCharcoalParticals_node” displayed in Figure 10 is used to build a linked list to generate the random cloud particles around the current location of touching point of stylus proxy on canvas. This simulates the air brush model in painting by spreading the pigment of color in a certain amount on canvas. The numbers of nodes generated are controlled by the density variables which calculate the cloud intensity
based on the physical properties set to medium and the pressure applied on to the canvas. The member variables “x”, “y” and “z” stores the location of cloud particles for particular node in this list. The variable “*nxt” points to the next node in the list.

The linked list generated from the structure “artistStroke_displayList_node” stores the display lists of all strokes applied on to the canvas, by converting all vertex information into display lists enhances the performance of the system. Each stroke when applied is recorded and stored in “artistStroke_node” and “record_artistStroke_node” linked lists simultaneously from start to end. Then all vertices in that stroke are converted to display lists followed by emptying the “artistStroke_node” to capture another stroke. The member variable “gStrokesDL” in this structure is to store the unique number of display list which is returned by the OpenGL API. The variable “*nxt” points to next node in the list.

The structure “artistCharcoalParticalsFall_node” displayed in the Figure 10 is used to store the animation sequence of the falling particles when dry medium is being applied to the canvas, the intensity of falling particles is directly proportional to the pressure applied on canvas. This simulates the artist forced stroke on canvas leaving some broken particles from the dry medium to fly down due to gravity. The member variables “x”, “y” and “z” stores the first occurrence of particles and the variables “updated_x”, “updated_y” and “updated_z” stores the current location after recalculation of height due to gravity. The member variable “lifeSpan” is to store the maximum amount of time that the particle has to be animated in order to cut the storage of particles to tradeoff with the performance. The member variable “midLife” is to store the threshold value to remain opaque before becoming transparent. The variable “updated_age” is
incremented for each loop of the program logic cycle. This gives the amount of time spent in animation. The rest variables “redComponent”, “greenComponent” and “blueComponent” represents the color of particles. The last variable “*nxt” points to the next node in the list. In due course of animation each particle’s age is noted and after crossing “midlife” value, its transparency is set to increases until its age reaches “lifeSpan” value, then the particular node holding that particle’s information is deleted from this list.

3.5.2 Class Diagrams

![Class Diagrams of 2D Canvas](image)

Figure 11: Class Diagrams of 2D Canvas.

This application has three classes namely “myCanvas”, “myCanvasFrame” and “myDrawingPen” as shown in Figure 11. Each class object is instantiated once during the program execution. The objects of classes “myCanvas”, “myDrawingPen” and “myCanvasFrame” holds the information about the canvas, drawing pen and canvas
frame material properties respectively. The application allows changing any properties during runtime.

3.5.3 State Diagrams

![State Diagram Showing Haptic Events in 2D Canvas](image_url)

The Figure 12 shows the state diagram of events generated in 2D canvas. Before starting drawing, the application has to be initialized to correct settings of the device and OpenGL, so as to create an environment to capture the artist strokes. The process of initializing device is done in separate routine, in this routine the device API such as hdInitDevice is called with enumerator value as HD_DEFAULT_DEVICE. This will initialize the default haptic device attached to the system. All events related to the device are captured with the help of API hlAddEventCallback, this API has a signature in which event id, object, event thread, and event callback procedure has to be passed as a
parameter. All events are distinguished by their event id’s and handled with the respective call back functions. These event signatures are added at the time of initializing device. The API, hdEnable with parameter as HD_FORCE_OUTPUT ables to capture the force feedback information. All objects which are needed to be recognised by haptic stylus have to be defined. Each object has its own id which is obtained by calling API hlGenShapes. This will generate an unique id for each object in haptic space. The device can detect the collision between objects and stylus and can have a specific face detection types either in front, back, or both front and back. There is an API called hlTouchableFace by passing parameter as HL_FRONT, HL_ BACK or HL_FRONT_AND_BACK can determine what type of surface involves when touching stylus with the object. The API hdStartScheduler will start the event loop for the haptic device, in this loop all events which are defined are detected and handled respectively.

As the device gets successfully initialized without any errors, It is ready for capturing the movements of its stylus. All stylus movements and its status attributes are stored in a linked lists. This linked list has to be initialized with null values before reading any data. All variables related to the drawing application have to be initialized with default values. The moment when the stylus proxy touches canvas, application enters into paint mode as shown in Figure 1. The canvas touching location is stored and sets the stroke’s count to 1 denoting the starting point of the stroke. When ever the stylus proxy untouches canvas it denotes the ending point of the stroke, the inbetween points of the stroke are recorded as per motion of the stylus. At each point in the stroke a random cloud points with different positional attributes are calculated to simulate the width of the stroke and this width is determined by the point size. The display lists are the fast way of
accessing graphic entities. When ever the stylus proxy is in still touch with canvas, all points and device attributes are recorded in to the linked list. This linked list can be increased to an unlimited size based on the stroke. As the stylus proxy untouches the canvas, all the positional and device status attributes are copied to another linked list and positional attributes are combined to form a display list. The linked lists are read and displayed according to the first come first serve basis.

3.5.4 Visualizing of Drawings

The data which is being displayed on to the canvas is to be refreshed according to the frame rate to get an animation effect, like drawing onto the canvas screen. All the data which is stored in different modes is retrieved and displayed in one cycle. The refreshing rate also depends on the number of graphic entities to be displayed onto the screen.

In normal display the drawing data is read from the linked lists and displayed as it is stored, that means the display lists and linked list are read and displayed on to the canvas and the current stroke having the data being recorded at the other end is read from front and displayed simultaneously. The trace of artist display mode displays the points in the stroke without any stroke width. Force on stroke display mode displays the points in various point size to illustrate the magnitude of the force applied for a particular point. Small force corresponds to the smaller point size and large force corresponds to bigger point size. The force visualization displays the artist’s strokes according to the magnitude, the stroke points are visualized in 3D space, having small magnitude point located in near side and large magnitude to the far end. It represents the mesh like structure in 3D space.
3.6 Design and Implementation of 2.5D Haptic Canvas

2.5D Art canvas is a slight modification of 2D Art canvas. This canvas supports the media, with which an artist can paint or sketch on to the canvas having cylindrical structures. This is actually a mix of 2D and 3D canvas. The canvas can be deformed into a “wavy” surface. The media which has semi fluid properties is simulated in this type of painting. The artist can perform painting activity by moving the stylus in any direction in the space allowing cylindrical structure to fall freely with a gravitational force. This creates a unique pattern on the uneven surface of the canvas.

Figure 13: Graphic Scene from 2.5D Canvas System

Figure 13(a) represents the canvas which can be deformed to a wavy surface. Figure 13(b) represents the complete process of painting applied to the 2.5D canvas, where the tubular structure is allowed to flow free fall and hitting on the surface of canvas.
3.6.1 Type defined structures

The Figure 14 displays different type defined structures used in the application. These structures are used as dynamic storage unit by implementing them as linked lists. The “artistStroke_node” structure stores all movements of stylus proxy while the application is in “painting_mode”. This linked list is set to null whenever the application turns into “normal_mode”. The data which is stored in a sequence of nodes consisting of location values are used to generate display lists with the help of other attributes in this structure. In this structure the member variables “x”, “y” and “z” are used to store location values of stylus proxy. The member variables “brush_gAngle_x”, “brush_gAngle_y” and “brush_gAngle_z” are used to store stylus proxy orientation in
space. The member variables “redComponent”, “greenComponent” and “blueComponent” are used to store RGB color values for that node. The variable “greyValue” is used to store the average RGB color value. This color attributes can be changed at any point of time while the application is in “painting mode”. The color of paint medium is determined by this attributes. The member variable “strokeCount” is used to store the count of strokes generated by the artist. The variable “touchGround” is declared as a Boolean value is used in simulating free fall. The variables “startPoint” and “endPoint” are the Boolean values which are set as stroke beginning and ending. The variables “touchedVertex_x”, “touchedVertex_y” and “touchedVertex_z” are used to store the location point when the paint medium touches the canvas. The member variable “radius” is used to store the radius of tube like structure. This radius can be dynamically changed in paint application. This denotes the thickness of medium. The member variables “mass” and “viscosity” are used to set the physical properties of paint medium. The pointer variables “*starting_node_of_artistPaintPArticals” stores the starting address of circular vertices and “*nxt” stores the address of next node in this list.

The structure “record_artistStroke_node” is used to copy the values of the structure “artistStroke_node”. This structure stores the activities of painting continuously, the “artistStroke_node” structure is used temporarily between strokes and set to null in process of freeing memory. The structure “artistStroke_displayList_node” is used to store the display lists generated in the process of painting. All display lists are stored as linked lists. For each stroke the application generates a separate linked list. The structure “artistPaintPArticals_node” is used to store the circular vertices around the location of proxy which is stored in structure “artistStroke_node”.

29
3.6.2 Class Diagrams

![Class Diagram of 2.5D Canvas]

Figure 15: Class Diagram of 2.5D Canvas

Figure 15 shows the only class used in this system to encapsulate the attributes of canvas. All members in this class are declared public and they can be accessed anywhere in the program. The member variables “width”, “height” and “depth” are used to store the dimensions of the canvas. These values are used in a function which is used to create the canvas. The member variables “screenLocation_x”, “screenLocation_y” and “screenLocation_z” are used to store the location of canvas in space. The member functions “createCanvas25D” and “displayCanvas25D” are used to create and display canvas using OpenGL APIs. The member function “displayCanvas25Dy” is used to display the wavy surface canvas. This canvas can change its surface elevation dynamically allowing the artist to create the wavy surface.
3.6.3 State Diagrams

The Figure 16 shows the state diagram of haptic events generated during the process of canvas deformation. The system may follow any cycle starting from initial node to final node of diagram. Each cycle has a specific set of tasks to be performed in a loop.
Figure 17(a), (b) Bottom 17(c), (d): Image Sequence Showing Canvas Deformation

Figure 17(a) shows the initial view of 2.5D canvas having flat surface. Figure 17(b, c, d) shows the deformation of the canvas creating wavy surfaces by collision detection through haptic stylus proxy.

3.6.4 Collision Detection in 2.5D Haptic Canvas

Consider the Figure 18, which represents a part of canvas vertices. At any point of time when stylus proxy touches the canvas the application turns into deformation mode. The canvas touching point and touchable face are registered. The touchable face determines the canvas elevation or depression. The canvas touching point is mapped to any four vertices based on nearest point. In Figure 19, the touching point is very near to
the blue vertex. The blue vertex is treated as collision vertex on canvas, every vertex centering the collided vertex has the shift in “z” value based on touchable face.

Figure 18: Mesh Diagram Showing Canvas Coordinates.

Figure 19: Mesh Diagram Showing Centered Collided Vertex and Surrounding Vertexes
the surrounding vertexes are elevated or depressed in less “z” value compare with inner region of vertices. This process creates smooth elevation and depression around point of collision.

3.6.5 Painting Process

![State Diagram of Haptic Events in Painting Mode]

Figure 20: State Diagram of Haptic Events in Painting Mode

The Figure 20 shows the state diagram of painting process, the application may follow any path from initial node to the final node handling all events and their call back processing during the course of program.

3.6.6 Design of Cylindrical Structures

When an application is in paint mode, the device is free to move in space over the canvas. Upon pressing the device stylus button 1 the stylus proxy locations are captured and stored simultaneously.
In Figure 21, the blue dot represents the captured location in space and black dots around the centered one represents the surface vertices of a tube-like structure. The number of black dots represents the resolution, the increased number of vertices looks more glossy and smooth texture and there is a tradeoff between the number of vertices and performance of the application.

All vertices are aligned in plane perpendicular to the stylus. This plane containing the surface vertices are oriented according to the stylus proxy angles. Figure 20 also shows a plane containing center vertex and the surrounding surface vertices in a plane this is almost perpendicular to the axis of stylus proxy. The event which is generated through the movement of stylus proxy is handled by “HapticEventMotionAnyObjectCallback” call back function. In this subroutine all the vertices are recorded and saved in linked list.

The orientation of this plane is determined by the angles of stylus proxy. Each vertex in the plane is processed individually to calculate the orientation values.
Figure 22: Tracking Vertex Planes According to the Stylus Movements

Figure 22 represents the record of two points in space by stylus. The upper plane is the newly recorded plane than the lower plane. The vertices of the planes are oriented according to the stylus proxy.

Figure 23: Group of Planes

Figure 23 shows the array of vertex planes arranged in a stock pile to get displayed using quadrilateral strip.
3.6.7 Surface Rendering

Each plane in memory is processed one by one.

![Diagram of surface rendering process](image)

Figure 24: Process of Generating Quad Strip using OpenGL API

Quadrilateral strip is rendered using OpenGL API `glBegin` with parameter as `GL_QUAD_STRIP` as to mention the topology. All vertices have to be mentioned in certain order to get the correct surface structure. In Figure 24 all vertices which are labelled with numbers are inserted into the OpenGL API in the ascending order to make sure each polygon is rendered perfectly.

The following is a program sequence for adding vertices into quad strip.

```
BEGIN(GL_QUAD_STRIP);
v1
v2
v3
...
vN
END();
```
Figure 25 shows the cylindrical structure which has many vertices are oriented in different directions are collectively displayed using quadrilateral strip. This gives the effect of painting media which is like a substance protruding out of canvas perpendicularly are also shown in Figure 26.

Figure 26: Cylindrical Structures after Surface Generation

3.6.8 Free Fall Simulation

Each vertex which belongs to paint medium is computed a new vertical height value, caused by gravitational pull and rendered simultaneously. A new height value is
assigned to give an animation in sequence of frames. This can be illustrated in Figure 27 and Figure 28.

![Figure 27: Illustration of Free Fall with Re-Orientation of Vertex Plane](image)

![Figure 28: Free Fall Simulation with Re-Orientation of Vertex Plane](image)

### 3.7 Design and Implementation of 3D Haptic Canvas

3D canvas is a complex canvas rendered using haptic stylus. The movements of stylus proxy are captured while artist tries to move the stylus in a free space. The texture of this canvas may range from simple plane like structure to a complex structure which is way beyond artist’s visualization. The canvas structure can be rendered symmetrically or
asymmetrically around certain pivot represented by stylus proxy. The canvas can take up to any shape which allows the artist to paint something on it. Symmetrical structure looks like a connected circular discs placed in a random fashion. Asymmetrical structure takes any random shape and size which are the output of random gesture of haptic stylus. The simplest structure can be rendered is as a pot. Pottery is a symmetrical structure. It can be centered at any axis like shown in Figure 29.

Figure 29: The 3D Canvas
3.7.1 Type Defined Structures

Figure 30: Type Defined Structures in 3D Haptic Canvas

Figure 30 shows the structures used for storing data by linked lists.
3.7.2 State Diagrams

![State Diagram](image)

Figure 31: State Diagram of Haptic Events in Creating Canvas

Figure 31 shows the state diagram of events occurred in designing the 3D canvas.

3.7.3 Rendering Canvas

There are two phases of rendering canvas onto screen, in first phase the program tries to render using previously created display lists. In second phase the program renders the part of canvas which is in while capturing mode, the performance of the system may not affected by the display lists but may effected by the current rendered vertices. The number vertices in current part of canvas determine the overall performance.

The haptic stylus proxy position is captured while the application is in building canvas mode. Each position captured is processed to get the distance from the center of axis to the current captured position. This distance is used as a radius to calculate circular vertices around the center of axis. This is illustrated in Figure 32 and 33.
In asymmetrical canvas, the circular vertices are captures around the new axis of rotation which is parallel inclined to the axis of stylus. This capturing is achieved by processing each vertex.

At each point of capture all circular vertices are stored continuously until the button 1 of stylus is released. As soon as the button 1 gets released a procedure is called to which creates display lists, this display list increases the overall system performance.
by rendering directly from display lists. This creates a circular structure around axis of rotation. The complexity of the structure depends on the number of captured proxy positions. The haptic shapes are not rendered while in capturing mode since it obstructs the path of stylus blocking between the surfaces already formed. This limits the free flow of stylus. Once the capturing is done and user satisfies then haptic rendering is done in order to introduce the feel of object presented in the virtual space as shown in Figure 34.

![Image](image.png)

Figure 34: Circle Asymmetrical 3D Canvas

### 3.7.4 Rendering Canvas Sketch

Canvas sketch is rendered only in the presence of canvas. in order to sketch there should be some base which captures the location of sketch. Once canvas is rendered in any fashion, It is released in order to get collided with haptic stylus proxy. Whenever the user tries to sketch on the so formed canvas the collision points are traced and processed to form a line segments. This line segment represents the stroke. Since the canvas object is not modified but a layer of line vertices are added forming line segments.
over the surface, this part adds the performance since the addition of line segments are a part from the base canvas geometry.

Figure 35: Illustration of Vertex Tracing.

Each stroke is processed individually, the tracing of vertices starts whenever the proxy touches the canvas and it continues to record all the vertices until user releases button 1. As soon as button 1 is released all the vertices are formed as line segments of a display lists. Each stroke has its own display lists. Figure 35 shows the vertex which is been captured is rendered as a part of line segment.
CHAPTER IV

CASE STUDIES

This project is about a software tool, which assists an artist to create her artwork with great flexibility. The device allows the artist to capture an artistic gesture which is in 3D space and transforms that into an artistic pattern on a virtual canvas. This virtual canvas can be felt through haptic senses. The software tool allows the artist to explore the dimensionality of canvas in 2D, 2.5D and 3D, each dimension giving different feel while creating artwork.

The project Haptic Enabled Multidimensional Canvas was designed and developed using OpenGL, OpenHaptics Toolkit and Microsoft Visual Studio 2008. This software tool was used and tested by artists Matthew Kolodziej and Eunsu Kang. Kolodziej is an associate professor at The University of Akron in the Department of Painting and Drawing. His work mainly focuses on the perception of archaeological sites establishing concrete relationships between the structures of fragments and residues in the sites. His recent paintings are made of gel medium. They express the information about structures and their deformation[29]. Kang is an assistant professor at The University of Akron in the Department of New Media. Kang’s work mainly focuses on hybridization of artwork using new media technologies {30}. Kang describes this
tool as a new media tool with which an artist can create 3D structural patterns with much flexibility without any constraints.

4.1 System Setup

The project device is connected to the computer system using 6-6 pin FireWire cable that exceeds the IEEE 1394 implementation recommendations. Figure 36 (a) shows the picture of FireWire cable.

![FireWire cable](a)

Figure 36: System Setup

Figure 36 (b) shows the computer FireWire port connected with the FireWire cable. Figure 36 (c) shows the back side of the device with power and FireWire cables plugged in. The haptic device can be calibrated by running a utility program supplied by Sensable Technologies. Figure 37 shows the GUI of a utility program which can be used to test and calibrate the device before using it.
4.2 Case Study I: Demonstration of 2D Haptic Canvas

This case gives a brief demonstration of using the software tool as a 2D canvas. Figure 39 shows the application graphical user interface having two different user panels which are designed to capture stylus proxy positions on the left side and displaying the system and device attributes on the right side. The window title shows the information
regarding the current frame rate and canvas case type. In this case Kolodziej tried to create a monumental stair case by using the device as a tool. This is illustrated using Figure 40. While using the software tool Kolodziej stated that, “as artist gets more used to the device, the software tool becomes more flexible in creating virtual arts” (personal communication, 3/25/2011).

Figure 39: 2D Haptic Canvas GUI

Figure 40: Software Testing by Kolodziej.
Figure 41. Monumental Stair Case by Kolodziej.

Figure 41 (a) shows the program snapshot of the artwork, the figure 41 (b) shows the snapshot of the tracking points of the stylus proxy and figure 41 (c) shows the snapshot of force resistance applied on the canvas, the amount of resistance is visualized by different opacity.

Figure 42: Individual Components of the Artwork.

Figure 42 (a), (b) shows the snap shot of zoomed in individual components of the artwork. Figure 42 (c) shows the cross section view of visualization of force resistance applied on the canvas while creating the artwork. This image also visualizes the depth of
penetration of the stylus proxy beyond the canvas surface. In this image the extreme left covers the points which are penetrated to the threshold level, the points to the extreme right are penetrated at surface level and are shown with low opacity to distinguish between force resistance values.

This software tool is designed to maintain optimum performance during the entire runtime. In order to create the artwork which is shown in figure 40, the program generates 108 display lists. All display lists are rendered with a frame rate of 42 frames per second which is considered a decent frame rate for an interactive graphic application.

4.3 Case Study II: Demonstration of 2.5D Haptic Canvas

This case gives a brief demonstration of using the software tool as a 2.5D canvas. Figure 43 shows the graphical user interface having one single user space which is designed to capture stylus proxy positions on, above and below the surface of a canvas. The window title bar shows the information regarding the current frame rate and canvas case type. Initially the canvas is represented as flat 2D canvas allowing the artist to create a “wavy” surface by touching either side of the canvas. The artist can feel the resistance from the canvas surface while creating a wavy surface. This resistance is proportional to the force applied by the stylus proxy on the surface. This interaction is used to deform the canvas at the touching point allowing the artist to generate a wavy surface.
Figure 43 shows the sequence of steps during the 2.5D canvas generation. Figure 44 (a) shows the surface elevation by the stylus proxy moving from bottom to top. The elevated height is rendered with different color to represent the resistance generated in the device. The application is programmed to produce more resistance as the elevation increases. There is a threshold value where the canvas surface cannot be elevated beyond that point. Figure 44 (b) shows the reverse direction creating a depression in the surface. Also this depression region is rendered using a range of colors to represent the resistance.
produced in the device. Figure 44 (c) shows the complete 2.5D canvas having certain elevations and depressions which resemble a wavy surface.

Figure 45 (a) shows the top view of such a 2.5D canvas. Figure 45 (b) and (c) show the painting process over the surface of the canvas. In this case a gel like medium is simulated in the painting process. The stylus proxy is tracked over the surface whenever an artist makes a gesture. The gel like medium is allowed to free fall from the traced points. This simulates the process of painting over 2.5 D canvases. In order to create the artwork shown in figure 45 (c), the program takes 12 display lists to render gel medium on the surface of the canvas. All display lists including the deformed 2.5D canvas are rendered with a frame rate of 40 frames per second which is considered a decent frame rate for an interactive graphic application.

4.4 Case Study III: Demonstration of 3D Haptic Canvas

This case gives a brief demonstration of using the software tool as 3D canvas. Figure 46 shows the graphical user interface having one single user space window which
is designed to capture the stylus proxy positions in a 3D space. The window title bar shows the information regarding the current frame rate and canvas case type and the number of vertices used to render geometrical structures. Initially the canvas is represented as an empty space allowing the artist to create any 3D structure by the movements of the stylus proxy in 3D space. The artist can feel the resistance from the surface of the structures. This resistance is proportional to the force applied by the stylus proxy on the surface. Kolodziej and Kang used and tested this part of the program. Kolodziej commented that this application may be used to view the abstract art in multiple angles. Kang stated that this application can be used to create artistic structures without any real world constraints. This 3D canvas not only allows the artist to create complex structures but also paint over the surface. The artist can also create artwork without any resistance by device by simply tracking the stylus proxy gesture in a 3D space.

Figure 46: 3D Haptic Canvas GUI
Figure 46 shows the graphical user interface for the 3D canvas, here the application is in canvas creation mode initially waiting to grasp the stylus proxy gestures in a 3D space to create complex structures. Device attributes are also displayed in GUI, which displays current proxy position, force resistance and opacity values. The virtual structures can be created either symmetrically around center pivot or free form. These complex structures are made to have a haptic sense, which locks the device stylus proxy at intersecting parts of the surface and also they are made without a haptic sense allowing free formation of structures removing all haptic constraints.

![Monumental Stair Case in 3D Canvas.](image)

Figure 47: Monumental Stair Case in 3D Canvas.

Figure 47 shows the artwork monumental stair case by Kolodziej in different angles. Here in this 3D canvas Kolodziej chooses the canvas object as a medium and the 3D space as a canvas. The gestures are traced to produce the artwork. In this application the canvas object can be manipulated with different opacities to render the parts of the canvas as transparent objects. He created this artwork by moving the stylus proxy in the 3D space simultaneously controlling the opacity value, resulting in the image produced in Figure 48.
Kang also used this application to generate complex patterns. The structures are formed using a single gesture as well as multiple gestures. Figure 49 and 50 shows the sequence of images produced during program execution.
Figure 50: Complex Structures II Rendered by Kang

Figure 51 shows the image sequence produced during program execution. These images are the examples of a single stroke gesture creating simple objects.

Figure 51: Simple Structures Rendered by Kang

This application also allows the artist to touch and drag over the surfaces to simulate the painting process over simple and complex structures. This part of the application is an extended work inspired from the works of the authors mentioned in the chapter I, a survey of the previous work section. Figure 52 shows the image sequence, where painting is being applied over various structures surfaces with the haptic sense.
This application also allows the artist to create art patterns in 3D space which further can be viewed at different angles. This case tool is used by me, the author of this thesis, to draw portraits which can be viewed at different angles. Figure 53 shows the same with three different angles of views. This portrait is drawn using the stylus proxy gesture in 3D space without a haptic sense having motion of the stylus in a resistance free space. The portrait or any other art pattern created so far can be archived in a system as data files. This file contains the device attribute information for every point traced. The sample case file is shown in figure 54. In this file the information regarding a point which is captured in 3D space is recorded as the stylus proxy positions, the stylus proxy rotational values, the opacity value and the
time stamp of a point. These attribute values are used in further analyses which are explained in the summary chapter of this thesis.

<table>
<thead>
<tr>
<th># Art DataFile Version 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINT_DATA</td>
</tr>
<tr>
<td>X Y Z V GA1 GA2 GA3 TCNT</td>
</tr>
<tr>
<td>999999,999999,999999,999999,999999,999999,999999,999999,</td>
</tr>
<tr>
<td>0.635902,0.429174,-0.216092,0.29,0.1,-0.263213,-1.24172,0.069090,049652960,</td>
</tr>
<tr>
<td>0.632619,0.429841,-1.25615,0.29,0.1,-0.245341,0.740499,0.659388,3.98552921,</td>
</tr>
<tr>
<td>0.617043,0.432998,-0.210713,0.29,0.1,-0.235114,0.740444,0.870676,3.48652828,</td>
</tr>
<tr>
<td>0.629486,0.430024,-0.213676,0.29,0.1,-0.241505,0.741172,0.870676,3.48652785,</td>
</tr>
</tbody>
</table>

Figure 54: Sample Art Data File.

This software tool is designed to maintain optimum performance during the entire runtime. In order to create the artwork which is shown in case study III, the program generates display lists which are rendered at an average frame rate of 40 frames per second which is considered a decent frame rate for an interactive graphic application. All these test cases are run on a system having 3 GB RAM and processor with a clock rate of 3.4 GHz.
CHAPTER V

SUMMARY AND FUTUREWORK

5.1 Summary

The project Haptic Enabled Multidimensional Canvas is designed and developed using OpenGL, OpenHaptics Toolkit and Microsoft Visual Studio 2008, Version 9.0 SP1 (Service Pack I). The entire application is written in C++ language. The project is designed and developed in three different cases and they are painting on 2D canvas, painting on 2.5D canvas and painting on 3D canvas. The 2D canvas is generally a virtual canvas having two dimensional properties, allowing the artist to create any artistic patterns. The artist can feel the sense of touch through the haptic device and can also feel the friction produced when drawing on to the canvas. The friction produced is directly proportional to the material medium and the canvas surface properties, these properties can be manipulated during course of drawing allowing the artist to study different kinds of material properties. This project simulates the use of dry media and helps in learning application of dry media on the canvas. The 2.5D canvas is a virtual canvas having uneven surface to illustrate the curvy and wavy surface, this surface serves as a challenge to the artist in creating any art form. The medium of material in this type of application is a simulation of thick paste, which is allowed to free fall over the surface of canvas. The artist can move the stylus over the canvas to create artistic patterns with the help of the
thick paste like medium. The 3D canvas allows artists to create art canvas in any circularly symmetrical shapes around the stylus position. The artist can vary the color and intensity values while creating the canvas which allows blending more interesting patterns. The canvas so made by artist can be allowed to draw some thing on to it, which will be used as a structure to study complex forms and its nature in three dimensional spaces. The artist is able to draw inside and outside of the surface of this complex structure. This allows the artist to study the painting styles over complex structures. Through this project the artist can self explore different dimensions of canvas and can have the experience of painting onto virtual canvas with the help of tactile feedback force from the haptic device.

5.2 Future Work

As the system is based on simulation of vertex values over time, the vertex simulation is very expensive over the graphics processing. Each vertex is stored and retrieved linearly which causes a reduced speed of rendering and ultimately reducing the frame rate, on the other hand high frame rate is required in interactive graphics. This will set a tradeoff between vertex and display lists. Though display lists increase the performance by storing in single location and decreasing rendering time, the display list cannot be used to manipulate the vertex dynamically. The issues related to the performance need to be addressed in future work.

Also there is another recommendation of future work on this project, as this project deals with capturing the hand movements and displaying the patterns simultaneously allowing the artist to perform interactive art simulations. These patterns
can be stored and analyzed to study the nature and style of one artist from others. This research can also establish the foundation of studying the individual patterns created by artist and can recognize a digital signature in the art.
REFERENCES


APPENDIX

The following is a source code of this project which is implemented in C++ language using OpenGL and OpenHaptics API’s. Each case study is implemented individually and the following are the corresponding header and cpp files.

Case I:

/***** Main Header File *****/
#include "wnd.h"
#include "openGL.h"
#include "constants.h"

/* Function prototypes. */
void exitHandler(void);
void drawSceneHaptics();
void drawSceneGraphics();
void drawCursor();
void updateWorkspace();
double determineFPS(void);
OpenGL Functions

void renderScene(void);
void drawVertex(double xPos, double yPos, double zPos);
void drawCone(double xPos, double yPos, double zPos);
void drawCube(double xPos, double yPos, double zPos);
void drawCylinder(double xPos, double yPos, double zPos);
void drawSphere(double xPos, double yPos, double zPos);
void drawCanvas();
void drawCanvasFrame();
void drawCanvasControlPanel();
void drawCanvasStatusPanel();
void drawCanvasSketch();
void drawCanvasSketchParticlesFall();
void updateCanvasSketch();
void captureArtistStrokes();
void createDisplayLists();
float calculateGreyValue(float value);
double calculate_zLoc_for_force(double fValue);
double calculate_artistStrokeCount(double count, double x,double y);
void drawGraphicRoom(double xPos, double yPos, double zPos);
void drawHapticRoom(double xPos, double yPos, double zPos);
void drawVTKRoom(double xPos, double yPos, double zPos);
void drawMesh(double xPos, double yPos, double zPos);
void drawColorCube();
void drawPolygon(int a,int b,int c,int d);
void glutDisplay(void);
void glutReshape(int width, int height);
void glutIdle(void);
void glutMenu(int);
void mouseCallback(int button, int state, int x, int y);
void motionCallback(int x, int y);
void pasivemotionCallback(int x, int y);
void keyboardCallback (unsigned char key, int x, int y);
void specialKeysCallback (int key, int x, int y);
void init(camera *cam);
void glutDisplay(void);
void overlayDisplay(void);
void glutIdle(void);
void initGL();
void initHL();
void initScene();
void drawSceneHaptics();
void drawSceneGraphics();
void drawCursor();
void drawCursor2(double x, double y, double z);
void drawDotPattern(double x, double y, double z);
void drawButton(double xPos, double yPos, double zPos, bool enable, char *str);
void drawHapticButton(double xPos, double yPos, double zPos);
void drawMetrics();
void updateWorkspace();
void writeToArtTrack();
void writeToArtTrack2();
void readFromArtTrack(char *fileName);
void populateAverageArtTrack();
void populateAverageArtTrack2();
void groupArtTrack();
void writeObjectPointsToFile();
void drawObject();
void readVTKfile();
void writeVTKfile();
void displayVTKfile();
double setMediumColor(int component, double value);
void resetApplication();

/*******************************************************************************
End Functions
*******************************************************************************/

/*******************************************************************************
Haptics Functions
*******************************************************************************/

void button1UpCallbackFun(unsigned int ShapeID);
void button2UpCallbackFun(unsigned int ShapeID);
void button1DownCallbackFun(unsigned int ShapeID);
void button2DownCallbackFun(unsigned int ShapeID);
void MotionCallback(unsigned int ShapeID);
void GraphicsCallback(void);
void startHaptics(void *arg);
void startDrawing(void *arg);
void initializeHaptics();

void HLCALLBACK hapticButton1DownCallback(HLenum event, HLuint object, HLenum thread, HLcache *cache, void *userdata);
void HLCALLBACK hapticButton1UpCallback(HLenum event, HLuint object, HLenum thread, HLcache *cache, void *userdata);
void HLCALLBACK hapticButton2DownCallback(HLenum event, HLuint object, HLenum thread, HLcache *cache, void *userdata);
void HLCALLBACK hapticButton2UpCallback(HLenum event, HLuint object, HLenum thread, HLcache *cache, void *userdata);
void HLCALLBACK hapticTouchAnyObjectCallback(HLenum event, HLuint object, HLenum thread, HLcache *cache, void *userdata);

void HLCALLBACK hapticUnTouchAnyObjectCallback(HLenum event, HLuint object, HLenum thread, HLcache *cache, void *userdata);

void HLCALLBACK hapticEventMotionAnyObjectCallback(HLenum event, HLuint object, HLenum thread, HLcache *cache, void *userdata);

HDCallbackCode HDCALLBACK SetSpringStiffnessCallback(void *pUserData);
HDCallbackCode HDCALLBACK AnchoredSpringForceCallback(void *pUserData);

HDSchedulerHandle gCallbackHandle = 0;

/***********************************************************************************
End Functions
***********************************************************************************/

/***** OpenGL Header File ******/

/**************************** OpenGL Headers Include files ****************************/

End

/**************************** OpenGL Headers Include files ****************************/

static GLuint gVTKDisplayList = 0;
static GLuint gArtStrokesDisplayList = 0;
static GLuint gDotPatternDisplayList = 0;
static GLuint gCursorDisplayList = 0;
static GLuint gCanvasDisplayList = 0;
static GLuint gCanvasFrameDisplayList = 0;
static GLuint gCanvasControlPanelDisplayList = 0;
static GLuint gCanvasStatusPanelDisplayList = 0;
static GLuint gArtistBrushForceVisualizationsDisplayList = 0;

static GLuint gMeshDisplayList = 0;
static GLuint gVolumeDisplayList = 0;
static double gCursorScale;

/**************************** OpenGL Headers Include files ****************************/

70
class objectCount
{
    public:

    double totalTracedVerticesInStroke;
    double totalVerticesInStroke;
    double totalTracedVerticesInDrawing;
    double totalVerticesInDrawing;
    double totalDisplayLists;
    double totalNumberOfStrokes;

    objectCount()
    {
        totalTracedVerticesInStroke = 0.0;
        totalVerticesInStroke = 0.0;
        totalTracedVerticesInDrawing = 0.0;
        totalVerticesInDrawing = 0.0;
        totalDisplayLists = 0.0;
        totalNumberOfStrokes = 0.0;
    }

};

class myCanvas
{

    bool modified;

    double canvasXPosition;
    double canvasYPosition;
    double canvasZPosition;

    double canvasWidth;
    double canvasHeight;
    double canvasDepth;

    double canvasSmoothness;
    double canvasAbsorption;

    double canvasRedComponent;
    double canvasGreenComponent;
    double canvasBlueComponent;

    double canvasStiffnessConstant;

    double canvasDampingConstant;
    double canvasStaticFrictionConstant;
    double canvasDynamicFrictionConstant;

    public:

    myCanvas()
    {

modified = false;

canvasXPosition = 0.0;
canvasYPosition = 0.0;
canvasZPosition = 0.0;

canvasWidth = 3.5;
canvasHeight = 2.4;
canvasDepth = 0.1;

canvasSmoothness = 0.5;
canvasAbsorption = 0.5;

canvasRedComponent = 1.0;
canvasGreenComponent = 1.0;
canvasBlueComponent = 1.0;

canvasStiffnessConstant = 0.7;

canvasDampingConstant = 0.1;
canvasStaticFrictionConstant = 0.2;
canvasDynamicFrictionConstant = 0.3;

myCanvas(double width, double height, double depth, double smoothness, double absorption, double rC, double gC, double bC, double stiffConstant, double dampingConstant, double sFriction, double dFriction)
{
    modified = false;
    canvasXPosition = 0.0;
canvasYPosition = 0.0;
canvasZPosition = 0.0;

canvasWidth = width;
canvasHeight = height;
canvasDepth = depth;

canvasSmoothness = smoothness;
canvasAbsorption = absorption;

canvasRedComponent = rC;
canvasGreenComponent = gC;
canvasBlueComponent = bC;

canvasStiffnessConstant = stiffConstant;

canvasDampingConstant = dampingConstant;
canvasStaticFrictionConstant = sFriction;
canvasDynamicFrictionConstant = dFriction;

}

void createCanvas();
void displayCanvas();
void setCanvasLocation(double xPos, double yPos, double zPos);
void setCanvasWidth(double value);
void setCanvasHeight(double value);
void setCanvasDepth(double value);
void setCanvasSmoothness(double value);
void setCanvasAbsorption(double value);
void setCanvasRedComponent(double value);
void setCanvasGreenComponent(double value);
void setCanvasBlueComponent(double value);
void setCanvasStiffnessConstant(double value);
void setCanvasDampingConstant(double value);
void setCanvasStaticFrictionConstant(double value);
void setCanvasDynamicFrictionConstant(double value);

double* getCanvasLocation();
double getCanvasWidth();
double getCanvasHeight();
double getCanvasDepth();
double getCanvasSmoothness();
double getCanvasAbsorption();
double getCanvasRedComponent();
double getCanvasGreenComponent();
double getCanvasBlueComponent();
double getCanvasStiffnessConstant();
double getCanvasDampingConstant();
double getCanvasStaticFrictionConstant();
double getCanvasDynamicFrictionConstant();

void myCanvas::createCanvas()
{
    double xPos = canvasXPosition;
    double yPos = canvasYPosition;
    double zPos = canvasZPosition;
    double xScale = canvasWidth;
    double yScale = canvasHeight;
    double zScale = canvasDepth;
    double rC = canvasRedComponent;
    double gC = canvasGreenComponent;
    double bC = canvasBlueComponent;

    if (!gCanvasDisplayList)
    {
        gCanvasDisplayList = glGenLists(1);
        glNewList(gCanvasDisplayList, GL_COMPILE);
        glPushMatrix();
        glTranslatef(xPos, yPos, zPos);
        glPushMatrix();
        glColor3f(rC, gC, bC);
        glScalef(xScale, yScale, zScale);
        glutSolidCube(1.0);
        glPopMatrix();
        glPopMatrix();
        glEndList();
    }
}
glPopMatrix();
glEndList();

}
}

void myCanvas::displayCanvas()
{
    glCallList(gCanvasDisplayList);
}

double* myCanvas::getCanvasLocation()
{
    double _array[3] = { canvasXPosition, canvasYPosition, canvasZPosition};
    return _array;
}

double myCanvas::getCanvasWidth()
{
    return canvasWidth;
}

double myCanvas::getCanvasHeight()
{
    return canvasHeight;
}

double myCanvas::getCanvasDepth()
{
    return canvasDepth;
}

double myCanvas::getCanvasSmoothness()
{
    return canvasSmoothness;
}

double myCanvas::getCanvasAbsorption()
{
    return canvasAbsorption;
}

double myCanvas::getCanvasRedComponent()
{
    return canvasRedComponent;
}

double myCanvas::getCanvasGreenComponent()
{
    return canvasGreenComponent;
}

double myCanvas::getCanvasBlueComponent()
{
    return canvasBlueComponent;
}

double myCanvas::getCanvasStiffnessConstant()
{
    return canvasStiffnessConstant;
}
double myCanvas::getCanvasDampingConstant()
{
    return canvasDampingConstant;
}

double myCanvas::getCanvasStaticFrictionConstant()
{
    return canvasStaticFrictionConstant;
}

double myCanvas::getCanvasDynamicFrictionConstant()
{
    return canvasDynamicFrictionConstant;
}

class myCanvasFrame : public myCanvas
{

    public:

        myCanvasFrame()
        {
        
        }

    void createCanvasFrame();
    void displayCanvasFrame();

};

void myCanvasFrame::createCanvasFrame()
{

    if (!gCanvasFrameDisplayList)
    {
        gCanvasFrameDisplayList = glGenLists(1);
        glNewList(gCanvasFrameDisplayList, GL_COMPILE);
        glPushMatrix();
            //glScalef(1.65,1.05,1.0);
            glScalef(1.0,1.05,1.0);
        glPopMatrix();
        glPushMatrix();
            glTranslatef(0.0, -1.2, 0.0);
        glPopMatrix();
    
    
}
void myCanvasFrame::displayCanvasFrame()
{
  glCallList(gCanvasFrameDisplayList);
}

class myDrawingPen
{
  bool modified;
  double drawingPenXPosition;
  double drawingPenYPosition;
  double drawingPenZPosition;
  double drawingPenWidth;
  double drawingPenHeight;
  double drawingPenDepth;
  double drawingPenHardness;
  double drawingPenRedComponent;
}
double drawingPenGreenComponent;
double drawingPenBlueComponent;

public:

myDrawingPen()
{
    modified = false;

drawingPenXPosition = 0.0;
drawingPenYPosition = 0.0;
drawingPenZPosition = 0.0;

drawingPenWidth = 0.5;
drawingPenHeight = 10.0;
drawingPenDepth = 1.0;

drawingPenHardness = 0.5;

drawingPenRedComponent = 0.5;
drawingPenGreenComponent = 0.5;
drawingPenBlueComponent = 0.5;
}

myDrawingPen(double width, double height, double depth, double hardness, double rC,
            double gC, double bC)
{
    modified = false;

drawingPenXPosition = 0.0;
drawingPenYPosition = 0.0;
drawingPenZPosition = 0.0;

drawingPenWidth = width;
drawingPenHeight = height;
drawingPenDepth = depth;

drawingPenHardness = hardness;
drawingPenRedComponent = rC;
drawingPenGreenComponent = gC;
drawingPenBlueComponent = bC;
}

void createDrawingPen();
void displayDrawingPen();

void setDrawingPenLocation(double xPos, double yPos, double zPos);
void setDrawingPenWidth(double value);
void setDrawingPenHeight(double value);
void setDrawingPenDepth(double value);
void setDrawingPenHardness(double value);
void setDrawingPenRedComponent(double value);
void setDrawingPenGreenComponent(double value);
void setDrawingPenBlueComponent(double value);
double* getDrawingPenLocation();
double getDrawingPenWidth();
double getDrawingPenHeight();
double getDrawingPenDepth();
double getDrawingPenHardness();
double getDrawingPenRedComponent();
double getDrawingPenGreenComponent();
double getDrawingPenBlueComponent();
}

void myDrawingPen::createDrawingPen()
{
    double xPos = drawingPenXPosition;
    double yPos = drawingPenYPosition;
    double zPos = drawingPenZPosition;
    double rC = drawingPenRedComponent;
    double gC = drawingPenGreenComponent;
    double bC = drawingPenBlueComponent;

    double kCursorRadius = drawingPenWidth;
    double kCursorHeight = drawingPenHeight;
    GLUquadricObj *qobj = 0;

    if (!gCursorDisplayList)
    {
        gCursorDisplayList = glGenLists(1);
        glNewList(gCursorDisplayList, GL_COMPILE);
        qobj = gluNewQuadric();
        gluTranslated(0.0, 0.0, kCursorRadius);
        glColor3f(rC, gC, bC);
        glutSolidSphere(kCursorRadius, RESOLUTION, RESOLUTION);
        gluCylinder(qobj, kCursorRadius, kCursorRadius, kCursorHeight, RESOLUTION, RESOLUTION);
        gluTranslated(0.0, 0.0, kCursorRadius + 9.5);
        glutSolidSphere(kCursorRadius, RESOLUTION, RESOLUTION);
        gluDeleteQuadric(qobj);
        glEndList();
    }
}

void myDrawingPen::displayDrawingPen()
{
    HLdouble proxyxform[16];
glPushAttrib(GL_CURRENT_BIT | GL_ENABLE_BIT | GL_LIGHTING_BIT);
glPushMatrix();
    hlGetDoublev(HL_PROXY_TRANSFORM, proxyxform);
    glMultMatrixd(proxyxform);
    glScaled(gCursorScale, gCursorScale, gCursorScale);
    glCallList(gCursorDisplayList);
    glPopMatrix();
    glPopAttrib();
}

double* myDrawingPen::getDrawingPenLocation()
{
    return _array;
}

double myDrawingPen::getDrawingPenWidth()
{
    return drawingPenWidth;
}

double myDrawingPen::getDrawingPenHeight()
{
    return drawingPenHeight;
}

double myDrawingPen::getDrawingPenDepth()
{
    return drawingPenDepth;
}

double myDrawingPen::getDrawingPenHardness()
{
    return drawingPenHardness;
}

double myDrawingPen::getDrawingPenRedComponent()
{
    return drawingPenRedComponent;
}

double myDrawingPen::getDrawingPenGreenComponent()
{
    return drawingPenGreenComponent;
}

double myDrawingPen::getDrawingPenBlueComponent()
{
    return drawingPenBlueComponent;
}
/*******************************************************************************
OpenGL variables
*******************************************************************************/
typedef struct cameras {
    float pos[3];
    float lookAt[3];
    float lookUp[3];
} camera;

struct artistDrawingMaterialAttributes{
    //double hardness;
    //double softness;
    bool redComponent;
    bool greenComponent;
    bool blueComponent;
} artDrwMtrAtr;

struct artistCharcoalParticals_node {
    double x;
    double y;
    double z;
    artistCharcoalParticals_node *nxt;
};
artistCharcoalParticals_node *start_artistCharcoalParticals_node = NULL;

struct artistCharcoalParticalsFall_node {
    double x;
    double y;
    double z;
    double updated_x;
    double updated_y;
    double updated_z;
    double lifeSpan;
    double midLife;
    double updated_age;
    double colorIntensity;
    double redComponent;
    double greenComponent;
    double blueComponent;
    artistCharcoalParticalsFall_node *nxt;
};
artistCharcoalParticalsFall_node *start_artistCharcoalParticalsFall_node = NULL;

struct artistStroke_node {
    double x;
}
double y;
double z;

double brush_gAngle_x;
double brush_gAngle_y;
double brush_gAngle_z;

double brushForce;

double brushVelocity_x;
double brushVelocity_y;
double brushVelocity_z;

double canvasTouchLocation_x;
double canvasTouchLocation_y;
double canvasTouchLocation_z;

double redComponent;
double greenComponent;
double blueComponent;
double greyValue;

double zDifference;
double brushMode;
double pointSize;
double strokeCount;

artistCharcoalParticals_node *starting_node_of_artistCharcoalParticals;
artistStroke_node *nxt;
};
artistStroke_node *start_artistStroke_node = NULL;

struct artistStroke_displayList_node
{
    int gStrokesDL;
    artistStroke_displayList_node *nxt;
};
artistStroke_displayList_node *start_artistStroke_displayList_node = NULL;

struct record_artistStroke_node
{
    double x;
    double y;
    double z;

    double brush_gAngle_x;
double brush_gAngle_y;
double brush_gAngle_z;

double brushForce;

double brushVelocity_x;
double brushVelocity_y;
double brushVelocity_z;
double canvasTouchLocation_x;
double canvasTouchLocation_y;
double canvasTouchLocation_z;

double redComponent;
double greenComponent;
double blueComponent;
double greyValue;
double zDifference;
double brushMode;
double pointSize;
double strokeCount;

record_artistStroke_node *nxt;
};
record_artistStroke_node *start_record_artistStroke_node = NULL;

struct node
{
    double x;
    double y;
    double z;
    double greyValue;
    node *nxt;
};
node *start_ptr = NULL;

struct ppnode
{
    double x;
    double y;
    double z;
    double greyValue;
    ppnode *nxt;
};
ppnode *pp_start_ptr = NULL;

struct surfacepaintnode
{
    double x;
    double y;
    double z;
    double greyValue;
    surfacepaintnode *nxt;
};
surfacepaintnode *spN_start_ptr = NULL;

struct trianglStripNode
{
    double x;
    double y;
    double y;
double z;
double greyValue;
trianglStripNode *nxt;
};
trianglStripNode *tSN_start_ptr = NULL;

struct quadStripNode
{
    double x;
    double y;
    double z;
    double greyValue;
    quadStripNode *nxt;
};
quadStripNode *qSN_start_ptr = NULL;

struct dsnode
{
    //static GLuint gStrokesDisplayList;
    int gStrokesDL;
    dsnode *nxt;
};
dsnode *ds_start_ptr = NULL;

GLuint gStrokesDisplayList = 0;

struct nodeToFile
{
    double x;
    double y;
    double z;
    double xG;
    double yG;
    double zG;
    double xF;
    double yF;
    double zF;
    double greyValue;
    double stressConstant;
    nodeToFile *nxt;
};
nodeToFile *nTF_start_ptr = NULL;

struct volumeNode
{
    double x;
    double y;
}
double z;
double greyValue;

volumeNode *nxt;
}
volumeNode *volumeNode_start_ptr = NULL;

static camera cam;

static int mainWindow;
static int border = 6, h=500, w=800;

int cursor_X = 0;
int cursor_Y = 0;

int old_cursor_X = 0;
int old_cursor_Y = 0;

int new_cursor_X = 0;
int new_cursor_Y = 0;

int programMode = 3;

GLfloat vertices[][3] = {{-1.0,-1.0,-1.0},
                         { 1.0,-1.0, 1.0},
                         {-1.0, 1.0,-1.0},
                         { 1.0, 1.0,-1.0},
                         {-1.0,-1.0, 1.0},
                         { 1.0,-1.0, 1.0},
                         { 1.0, 1.0, 1.0},
                         {-1.0, 1.0, 1.0}};

GLfloat normals[][3] = {{-1.0,-1.0,-1.0},
                        { 1.0,-1.0, 1.0},
                        {-1.0, 1.0,-1.0},
                        { 1.0, 1.0,-1.0},
                        {-1.0,-1.0, 1.0},
                        { 1.0,-1.0, 1.0},
                        { 1.0, 1.0, 1.0},
                        {-1.0, 1.0, 1.0}};

//GLfloat colors[][3] = {{0.0,0.0,0.0},
//                        {1.0,0.0,0.0},
//                        {0.0,1.0,0.0},
//                        {0.0,0.0,1.0},
//                        {1.0,0.0,1.0},
//                        {1.0,1.0,1.0},
//                        {0.0,1.0,1.0}};

GLfloat colors[][3] = {{0.7,0.7,0.7},
                       {0.7,0.7,0.7},
                       {0.7,0.7,0.7},
                       {0.7,0.7,0.7},
                       {0.7,0.7,0.7},
                       {0.7,0.7,0.7},
                       {0.7,0.7,0.7},
                       {0.7,0.7,0.7}};
// GL_LIGHT0: the white light emitting light source
// Create light components for GL_LIGHT0
float ambientLight0[] = { 1.0f, 1.0f, 1.0f, 1.0f };  
float diffuseLight0[] = { 0.8f, 0.8f, 0.8f, 1.0f };  
float specularLight0[] = { 0.5f, 0.5f, 0.5f, 1.0f };  
float position0[] = { -10.0f, 5.0f, -10.0f, 1.0f };  

// GL_LIGHT1: the red light emitting light source
// Create light components for GL_LIGHT1
float ambientLight1[] = { 1.0f, 0.0f, 0.0f, 1.0f };  
float diffuseLight1[] = { 1.0f, 0.0f, 0.0f, 1.0f };  
float specularLight1[] = { 1.0f, 0.0f, 0.0f, 1.0f };  
float position1[] = { 10.0f, 5.0f, -10.0f, 1.0f };  

// GL_LIGHT1: the red light emitting light source
// Create light components for GL_LIGHT1
float ambientLight2[] = { 0.0f, 0.0f, 1.0f, 1.0f };  
float diffuseLight2[] = { 0.0f, 0.0f, 1.0f, 1.0f };  
float specularLight2[] = { 0.0f, 0.0f, 1.0f, 1.0f };  
float position2[] = { 10.0f, 5.0f, 10.0f, 1.0f };  

float xDir = 0.0;  
float yDir = 0.0;  
float zDir = 0.0;  

int transp_color;

std::string tracedPoints = "";
std::string objectPoints = "";

char tracePoints[200];

float viewAngle = 0.0;  
float tippAngle = 0.0;  
float elevationCanvas = 0.0;

float paintR = 0.8;  
float paintG = 0.8;  
float paintB = 0.8;

int *matrix3D;  
int matrixDimX = 0;  
int matrixDimY = 0;  
int matrixDimZ = 0;  
int choice_of_data_display = 1;

double surfaceArea = 0.0;  
double zoom = 1.0;  
double zoomScene = 0.0;
double lookPosition = 1.0;
double rotationAngle = 0.0;
double sceneRotationAngleX = 0.0;
double sceneRotationAngleY = 0.0;
double sceneRotationAngleZ = 0.0;
double canvasZ = -42.0;
double rad2deg = PI/180;
double radiusXYZ = 0.5;
double canvasTouchZ = 0.0;
double artistStrokeCount = 0.0;
double artistStrokeBrushSize = 1.0;
double artistCanvas_Z_Position = 0.0;
double artistStrokes_viewRotation = 0.0;

double greyVal = 0.0;
bool surfaceTriangles = false;
bool surfaceVertex = true;
bool randomVertex = true;
bool surfacePaint = false;
bool objectTouched = false;
bool firstPoint = true;
bool canvasPaint = false;
bool artistStrokeTraceLocations = false;
bool artistStrokeBrushLocations = true;
bool artistStrokeForceLocations = false;
bool artistStrokeAngleLocations = false;
bool artistStrokeVisualizations = false;

myCanvas *twoDCanvas;
myCanvasFrame *twoDCanvasFrame;
myDrawingPen *twoDDrawingPen;
objectCount *twoDOBJECTCOUNT;
/******************************************************************************/
OpenHaptics Headers Include files

*******************************************************************************

//include <QHHeadersWin32.h>
//include <QHHeadersGLUT.h>
#include <HD/hd.h>
#include <HDU/hduVector.h>
#include <HDU/hduError.h>
#include <HDU/hduMatrix.h>
#include <HDU/hduQuaternion.h>
#include <HDU/hduGenericMatrix.h>
#include <HDU/hduHapticDevice.h>

#include <HL/hl.h>
#include <HLU/hlu.h>

*******************************************************************************

End

*******************************************************************************

Haptics variables

*******************************************************************************

hduVector3Dd m_devicePosition;
bool omniButton1Down = false;
bool omniButton1Up   = false;
bool omniButton2Down = false;
bool omniButton2Up   = false;
hduVector3Dd proxy;
hduVector3Dd proxyRot;
hduVector3Dd deviceReactionForce;

hduVector3Dd canvasTouchCoordinates;

hduVector3Dd omniStylusPosition;

hduVector3Dd omniStylusPositionOld;

hduVector3Dd omniStylusGimbalAngles;

HDdouble omniForce[3];

double xOmniPosition = 0.0;
double yOmniPosition = 0.0;
double zOmniPosition = 0.0;

double xGimbal = 0.0;
double yGimbal = 0.0;
double zGimbal = 0.0;

double xGimbalinDegrees = 0.0;
double yGimbalinDegrees = 0.0;
double zGimbalinDegrees = 0.0;

int x;

int y;

int z;

/* Haptic device and rendering context handles. */
static HHD ghHD = HD_INVALID_HANDLE;
static HHLRC ghHLRC = 0;

/* Shape id for shape we will render haptically. */
HLuint gCanvasShapeId;
HLuint gCanvasFrameShapeId;
HLuint gCanvasControlPanelShapeId;
HLuint gCanvasStatusPanelShapeId;

HLuint gSphereShapeId2;

//const double kPI = 3.141593;
//const double kFovY = 40;
bool paintCanvas = false;
bool startRecording = false;
bool paintPottery = false;

bool artistButton = true;
bool studentButton = false;
//bool bRenderForce = false;

static HDdouble gSpringStiffness = 0.5;
static HDdouble gMaxStiffness = 1.0;
static HDdouble gStiffnessIncrement = 0.025;

/******************************************************************************/
#include <conio.h>
#include <stdio.h>
#include <assert.h>
#include <process.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sys/types.h>
#include <string>
#include <ctime>
#include <ctime>
#include <time.h>
#include <corona.h>

#if defined(WIN32)
#include <windows.h>
#endif
***** constant header file *****

#define NUMBEROFVOXELS 15
#define CURSOR_SIZE_PIXELS 20
#define RESOLUTION 32

#define WINDOW_WIDTH 1200
#define WINDOW_HEIGHT 900
#define WINDOW_POSITION_X 10
#define WINDOW_POSITION_Y 10

#define PI 3.14159265
//3.1415926535897932384626433832795

#define POINTSIZE 1.0

#define NORMAL_DATA_DISPLAY 1
#define TRACE_DATA_DISPLAY 2
#define FORCE_DATA_DISPLAY 3
#define ANGLE_DATA_DISPLAY 4
#define VISUAL_DATA_DISPLAY 5

#define RED 1
#define GREEN 2
#define BLUE 3
#define GREY 4
Case II:

/*****Main Header file *****/
#include "wnd.h"
#include "openGL.h"
#include "openHaptics.h"
#include "constants.h"

/* Function prototypes. */
void exitHandler(void);
void drawSceneHaptics();
void drawSceneGraphics();
void drawCursor();
void updateWorkspace();
double determineFPS(void);

nosti*******************************************************************************
OpenGL Functions
*******************************************************************************
void renderScene(void);
void drawVertex(double xPos, double yPos, double zPos);
void drawCone(double xPos, double yPos, double zPos);
void drawCube(double xPos, double yPos, double zPos);
void drawCylinder(double xPos, double yPos, double zPos);
void drawSphere(double xPos, double yPos, double zPos);
void draw3DRoom();
void drawCanvas();
void drawCanvasFrame();
void drawCanvasControlPanel();
void drawCanvasStatusPanel();
void drawCanvasSketch();
void drawTwoHalfDimCanvas();
void drawWaterParticles();
void createWaterParticlesPath(double xPos, double yPos, double zPos);
void updateWaterParticlesPath();
void updateCanvasSketch();
void updateCanvas();
void captureArtistStrokes();
void createDisplayLists();
float calculateGreyValue(float value);
double calculate_zLoc_for_force(double fValue);
double calculate_artistStrokeCount(double count, double x,double y);
void drawGraphicRoom(double xPos, double yPos, double zPos);
void drawHapticRoom(double xPos, double yPos, double zPos);
void drawVTKRoom(double xPos, double yPos, double zPos);
void drawMesh(double xPos, double yPos, double zPos);
void drawColorCube();
void drawPolygon(int a,int b,int c,int d);

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

void mouseCallback(int button, int state, int x, int y);
void motionCallback(int x, int y);
void passivemotionCallback(int x, int y);
void keyboardCallback (unsigned char key, int x, int y);
void specialKeysCallback (int key, int x, int y);
void init(camera *cam);

void glutDisplay(void);
void overlayDisplay(void);
void glutIdle(void);
void initGL();
void initHL();
void initScene();
void drawSceneHaptics();
void drawSceneGraphics();
void drawCursor();
void drawCursor2(double x,double y,double z);
void drawDotPattern(double x,double y,double z);
void drawButton(double xPos, double yPos, double zPos, bool enable, char *str);
void drawHapticButton(double xPos, double yPos, double zPos);
void drawMetrics();
void updateWorkspace();
void writeToArtTrack();
void writeToArtTrack2();
void readFromArtTrack(char *fileName);
void populateAverageArtTrack();
void populateAverageArtTrack2();
void groupArtTrack();
void writeObjectPointsToFile();
void drawObject();
void readVTKfile();
void writeVTKfile();
void displayVTKfile();
void updatePhysics();

*******************************************************************************
End Functions
*******************************************************************************

*******************************************************************************
Haptics Functions
*******************************************************************************

void button1UpCallbackFun(unsigned int ShapeID);
void button2UpCallbackFun(unsigned int ShapeID);
void button1DownCallbackFun(unsigned int ShapeID);
void button2DownCallbackFun(unsigned int ShapeID);
void MotionCallback(unsigned int ShapeID);
void GraphicsCallback(void);
void startHaptics(void *arg);
void startDrawing(void *arg);
void initializeHaptics();

void HLCALLBACK hapticButton1DownCallback(HLenum event, HLuint object, HLenum thread,
HLcache *cache, void *userdata);
void HLCALLBACK hapticButton1UpCallback(HLenum event, HLuint object, HLenum thread, HLcache
*cache, void *userdata);
void HLCALLBACK hapticButton2DownCallback(HLenum event, HLuint object, HLenum thread,
HLcache *cache, void *userdata);
void HLCALLBACK hapticButton2UpCallback(HLenum event, HLuint object, HLenum thread, HLcache
*cache, void *userdata);
void HLCALLBACK hapticTouchAnyObjectCallback(HLenum event, HLuint object, HLenum thread,
HLcache *cache, void *userdata);
void HLCALLBACK hapticUnTouchAnyObjectCallback(HLenum event, HLuint object, HLenum thread,
HLcache *cache, void *userdata);
void HLCALLBACK hapticEventMotionAnyObjectCallback(HLenum event, HLuint object, HLenum thread,
HLcache *cache, void *userdata);
HDCALLBACK HDCallbackCode SetSpringStiffnessCallback(void *pUserData);
HDCALLBACK HDCallbackCode AnchoredSpringForceCallback(void *pUserData);

HDSchedulerHandle gCallbackHandle = 0;

End Functions

******************************************************************************
End Functions
******************************************************************************
/******************************OpenGL Headers Include files******************************/

#include "constants.h"

/**********************************************************************
End
**********************************************************************/

/******************************OpenGL variables******************************/

typedef struct cameras {
    float pos[3];
    float lookAt[3];
    float lookUp[3];
} camera;

struct artistPaintParticals_node
{
    double index;
    double x;
    double y;
    double z;
    double updated_x;
    double updated_y;
}
double updated_z;

double viscosity;
double mass;

bool facingGround;

artistPaintParticals_node *nxt;
};
artistPaintParticals_node *start_artistPaintParticals_node = NULL;

struct artistWaterParticals_node
{

double x;
double y;
double z;

double updated_x;
double updated_y;
double updated_z;

double updated_path_deviation;

double lifeSpan;
double midLife;
double updated_age;
double colorIntensity;
double redComponent;
double greenComponent;
double blueComponent;

artistWaterParticals_node *nxt;
};
artistWaterParticals_node *start_artistWaterParticals_node = NULL;

struct artistWaterParticalsPath_node
{
    double x;
    double y;
    double z;
    artistWaterParticalsPath_node *nxt;
    artistWaterParticalsPath_node *pre;
};
artistWaterParticalsPath_node *start_artistWaterParticalsPath_node = NULL;
artistWaterParticalsPath_node *end_artistWaterParticalsPath_node = NULL;

struct artistStroke_node
{
    double index;
    double x;
    double y;
    double z;
double brush_gAngle_x;
double brush_gAngle_y;
double brush_gAngle_z;

double brushForce;

double brushVelocity_x;
double brushVelocity_y;
double brushVelocity_z;

double canvasTouchLocation_x;
double canvasTouchLocation_y;
double canvasTouchLocation_z;

double greyValue;
double redComponent;
double greenComponent;
double blueComponent;

double strokeCount;

bool touchGround;
bool startPoint;
bool endPoint;

double touchedVertex_x;
double touchedVertex_y;
double touchedVertex_z;
double radius;
artistPaintParticals_node *starting_node_of_artistPaintParticals;
artistStroke_node *nxt;
};
artistStroke_node *start_artistStroke_node = NULL;

struct artistStroke_displayList_node
{
    int gStrokesDL;
    artistStroke_displayList_node *nxt;
};
artistStroke_displayList_node *start_artistStroke_displayList_node = NULL;

struct record_artistStroke_node
{
    double x;
    double y;
    double z;
    double brush_gAngle_x;
    double brush_gAngle_y;
    double brush_gAngle_z;
    double brushForce;
    double brushVelocity_x;
    double brushVelocity_y;
    double brushVelocity_z;
double canvasTouchLocation_x;
double canvasTouchLocation_y;
double canvasTouchLocation_z;

double greyValue;

double radius;
double pointSize;
double strokeCount;

record_artistStroke_node *nxt;
};
record_artistStroke_node *start_record_artistStroke_node = NULL;

struct node
{
    double x;
    double y;
    double z;
    double greyValue;
    node *nxt;
};
node *start_ptr = NULL;

struct ppnode
{
    double x;
    double y;
double z;
double greyValue;
ppnode *nxt;
};
ppnode *pp_start_ptr = NULL;

struct surfacepaintnode
{
  double x;
  double y;
  double z;
  double greyValue;
  surfacepaintnode *nxt;
};
surfacepaintnode *spN_start_ptr = NULL;

struct trianglStripNode
{
  double x;
  double y;
  double z;
  double greyValue;
  trianglStripNode *nxt;
};
trianglStripNode *tSN_start_ptr = NULL;
struct quadStripNode
{
    double x;
    double y;
    double z;
    double greyValue;
    quadStripNode *nxt;
};
quadStripNode *qSN_start_ptr = NULL;

struct dsnode
{
    //static GLuint gStrokesDisplayList;
    int gStrokesDL;
    dsnode *nxt;
};
dsnode *ds_start_ptr = NULL;

GLuint gStrokesDisplayList = 0;

struct nodeToFile
{
    double x;
double y;
double z;
double xG;
double yG;
double zG;
double xF;
double yF;
double zF;
double greyValue;
double stressConstant;
	nodeToFile *nxt;
};
nodeToFile *nTF_start_ptr = NULL;

struct volumeNode
{

double x;
double y;
double z;
double greyValue;

*nxt;
};
volumeNode *volumeNode_start_ptr = NULL;
static camera cam;

static int mainWindow;
static int border = 6, h=500, w=800;

int cursor_X = 0;
int cursor_Y = 0;

int old_cursor_X = 0;
int old_cursor_Y = 0;

int new_cursor_X = 0;
int new_cursor_Y = 0;

int programMode = 3;

GLfloat canvasVertex[100][100][3];

GLfloat vertices[][3] = {{-1.0,-1.0,-1.0},
{ 1.0,-1.0,-1.0},
{ 1.0, 1.0,-1.0},
{-1.0, 1.0,-1.0},}
GLfloat normals[][3] = {{-1.0, -1.0, 1.0},
                      {1.0, -1.0, 1.0},
                      {1.0, 1.0, 1.0},
                      {-1.0, 1.0, 1.0}};

GLfloat colors[][3] = {{0.0, 0.0, 0.0},
                       {1.0, 0.0, 0.0},
                       {1.0, 1.0, 0.0},
                       {0.0, 1.0, 0.0},
                       {0.0, 0.0, 1.0},
                       {1.0, 0.0, 1.0},
                       {1.0, 1.0, 1.0},
                       {0.0, 1.0, 1.0},
                       {0.0, 0.0, 1.0},
                       {1.0, 0.0, 1.0},
                       {1.0, 1.0, 1.0}};
//
// {1.0,1.0,1.0},
//
// {0.0,1.0,1.0});

GLfloat colors[][3] = {{0.7,0.7,0.7},
                      {0.7,0.7,0.7},
                      {0.7,0.7,0.7},
                      {0.7,0.7,0.7}};

// GL_LIGHT0: the white light emitting light source
// Create light components for GL_LIGHT0
float ambientLight0[] = { 1.0f, 1.0f, 1.0f, 1.0f };  
float diffuseLight0[] = { 0.8f, 0.8f, 0.8f, 1.0f };  
float specularLight0[] = { 0.5f, 0.5f, 0.5f, 1.0f };  
float position0[] = { -10.0f, 5.0f, -10.0f, 1.0f };  

// GL_LIGHT1: the red light emitting light source
// Create light components for GL_LIGHT1
float ambientLight1[] = { 1.0f, 0.0f, 0.0f, 1.0f };  
float diffuseLight1[] = { 1.0f, 0.0f, 0.0f, 1.0f };  
float specularLight1[] = { 1.0f, 0.0f, 0.0f, 1.0f };  
float position1[] = { 10.0f, 5.0f, -10.0f, 1.0f };  

// GL_LIGHT1: the red light emitting light source
// Create light components for GL_LIGHT1
float ambientLight2[] = { 0.0f, 0.0f, 1.0f, 1.0f };
float diffuseLight2[] = { 0.0f, 0.0f, 1.0f, 1.0f };
float specularLight2[] = { 0.0f, 0.0f, 1.0f, 1.0f };
float position2[] = { 10.0f, 5.0f, 10.0f, 1.0f };

float xDir = 0.0;
float yDir = 0.0;
float zDir = 0.0;

int transp_color;

static GLuint gVTKDisplayList = 0;
static GLuint gArtStrokesDisplayList = 0;
static GLuint gDotPatternDisplayList = 0;
static GLuint gCursorDisplayList = 0;
static GLuint gCanvasDisplayList = 0;
static GLuint gCanvasFrameDisplayList = 0;
static GLuint gCanvasControlPanelDisplayList = 0;
static GLuint gCanvasStatusPanelDisplayList = 0;
static GLuint gArtistBrushForceVisualizationsDisplayList = 0;

static GLuint gMeshDisplayList = 0;
static GLuint gVolumeDisplayList = 0;

static double gCursorScale;

std::string tracedPoints = "";
std::string objectPoints = "";

char tracePoints[200];
float viewAngle = 0.0;
float tippAngle = 0.0;
float elevationCanvas = 0.0;

float paintR = 0.8;
float paintG = 0.8;
float paintB = 0.8;

int *matrix3D;
int matrixDimX = 0;
int matrixDimY = 0;
int matrixDimZ = 0;
int choice_of_data_display = 1;

double surfaceArea = 0.0;
double zoom = 1.0;

double zoomScene = 0.0;
double lookPosition = 1.0;
double rotationAngle = 0.0;
double sceneRotationAngleX = 0.0;
double sceneRotationAngleY = 0.0;
double sceneRotationAngleZ = 0.0;
double canvasZ = -42.0;
double rad2deg = PI/180;
double radiusXYZ = 0.5;
double canvasTouchZ = 0.0;
double artistStrokeCount = 0.0;
double artistStrokeBrushSize = 1.0;
double artistCanvas_Z_Position = 0.0;
double artistStrokes_viewRotation = 0.0;
double colorIntensity = 1.0;

/////
double artistCanvas_roll_angle = 0.0;
double artistCanvas_pitch_angle = 0.0;

double shift = 1.0;
/////

double greyVal = 0.0;
bool surfaceTrianlges = false;
bool surfaceVertex = true;
bool randomVertex = true;
bool surfacePaint = false;
bool objectTouched = false;
bool firstPoint = true;
bool firstCircle = true;
bool canvasPaint = false;
bool canvasUpdate = false;
bool artistStrokeTraceLocations = false;
bool artistStrokeBrushLocations = true;
bool artistStrokeForceLocations = false;
bool artistStrokeAngleLocations = false;
bool artistStrokeVisualizations = false;
bool startPaint = false;
bool endPaint = false;
class canvas25D
{
    public:
        double width;
        double height;
        double depth;
        double sceneLocation_x;
        double sceneLocation_y;
        double sceneLocation_z;
        double inclined_x;
        double inclined_y;
        double inclined_z;

canvas25D()
{
    width = 3.0;
    height = 1.5;
    depth = 0.01;
}
n_x = 0.0;

n_y = -1.0;

n_z = 0.0;

inclined_x = 90.0;

inclined_y = 0.0;

inclined_z = 0.0;
}

void createCanvas25D()

{

if (!gCanvasDisplayList)

{

ayList = glGenLists(1);

CanvasDisplayList, GL_COMPILE);

();

gCanvasDisplayList = glGenLists(1);

glNewList(gCanvasDisplayList, GL_COMPILE);

glPushMatrix();

glPopMatrix();

return;

}

return;

}
glRotatef(incl_x,1.0,0.0,0.0);

glRotatef(incl_y,0.0,1.0,0.0);

glRotatef(incl_z,0.0,0.0,1.0);

0.0, 0.0, 0.0);

();

1.0,1.0);

h, height, depth);

();

e(1.0);

());

); glPopMatrix();

); glPopMatrix();

); glPopMatrix();

); glEndList();

} 

}
void displayCanvas25D()
{
    glPushMatrix();
    glRotatef(sceneRotationAngleY,0.0,1.0,0.0);
    glRotatef(sceneRotationAngleX,1.0,0.0,0.0);
    glTranslatef(sceneLocation_x, sceneLocation_y, sceneLocation_z);
    glCallList(gCanvasDisplayList);
    glPopMatrix();
}

displayCanvas25Dy()
{
    glPushMatrix();
    glRotatef(sceneRotationAngleY,0.0,1.0,0.0);
    glRotatef(sceneRotationAngleX,1.0,0.0,0.0);
    //glTranslatef(sceneLocation_x, -0.3, sceneLocation_z);
    glPopMatrix();
}
glPushMatrix();

for(int i=0;i<99;i++)
for(int j=0;j<99;j++)
{

double k = - canvasVertex[i][j][1];

glColor4f(k,1.0 - k,k - 1.0,colorIntensity);

glBegin(GL_TRIANGLES);

 glNormal3f(canvasVertex[i][j][0],canvasVertex[i][j][1],canvasVertex[i][j][2]);

 glVertex3f(canvasVertex[i][j][0],canvasVertex[i][j][1],canvasVertex[i][j][2]);

 glNormal3f(canvasVertex[i][j+1][0],canvasVertex[i][j+1][1],canvasVertex[i][j+1][2]);

 glVertex3f(canvasVertex[i][j+1][0],canvasVertex[i][j+1][1],canvasVertex[i][j+1][2]);

 glNormal3f(canvasVertex[i+1][j+1][0],canvasVertex[i+1][j+1][1],canvasVertex[i+1][j+1][2]);

 glVertex3f(canvasVertex[i+1][j+1][0],canvasVertex[i+1][j+1][1],canvasVertex[i+1][j+1][2]);

}

}
glEnd();

glBegin(GL_TRIANGLES);

glNormal3f(canvasVertex[i][j][0], canvasVertex[i][j][1], canvasVertex[i][j][2]);

glVertex3f(canvasVertex[i][j][0], canvasVertex[i][j][1], canvasVertex[i][j][2]);

glNormal3f(canvasVertex[i+1][j+1][0], canvasVertex[i+1][j+1][1], canvasVertex[i+1][j+1][2]);

glVertex3f(canvasVertex[i+1][j+1][0], canvasVertex[i+1][j+1][1], canvasVertex[i+1][j+1][2]);

glNormal3f(canvasVertex[i+1][j][0], canvasVertex[i+1][j][1], canvasVertex[i+1][j][2]);

glVertex3f(canvasVertex[i+1][j][0], canvasVertex[i+1][j][1], canvasVertex[i+1][j][2]);

glEnd();

}  

}  

glPopMatrix();

};

}  

}  

}
canvas2D *myCanvas2D;

/******************************************************************************/
OpenHaptics Headers Include files

/******************************************************************************
******
OpenHaptics Headers Include files

/******************************************************************************

#include <QHHeadersWin32.h>
#include <QHHeadersGLUT.h>
#include <HD/hd.h>
#include <HDU/hduVector.h>
#include <HDU/hduError.h>
#include <HDU/hduMatrix.h>
#include <HDU/hduQuaternion.h>
#include <HDU/hduGenericMatrix.h>
#include <HDU/hduHapticDevice.h>

#include <HL/hl.h>
#include <HLU/hlu.h>

/******************************************************************************
******
Haptics variables

/******************************************************************************

hduVector3Dd m_devicePosition;

bool omniButton1Down = false;
bool omniButton1Up   = false;
bool omniButton2Down = false;
bool omniButton2Up = false;

hduVector3Dd proxy;

hduVector3Dd proxyRot;

hduVector3Dd deviceReactionForce;

hduVector3Dd canvasTouchCoordinates;

hduVector3Dd omniStylusPosition;

hduVector3Dd omniStylusPositionOld;

hduVector3Dd omniStylusGimbalAngles;

HDdouble omniForce[3];

double xOmniPosition = 0.0;

double yOmniPosition = 0.0;

double zOmniPosition = 0.0;

double xGimbal = 0.0;

double yGimbal = 0.0;

double zGimbal = 0.0;

double xGimbalinDegrees = 0.0;

double yGimbalinDegrees = 0.0;

double zGimbalinDegrees = 0.0;

int x;

int y;

int z;

/* Haptic device and rendering context handles. */

static HHD ghHD = HD_INVALID_HANDLE;

static HHLRC ghHLRC = 0;
/* Shape id for shape we will render haptically. */
HLuint gCanvasShapeId;
HLuint gCanvasFrameShapeId;
HLuint gCanvasControlPanelShapeId;
HLuint gCanvasStatusPanelShapeId;
HLuint gSphereShapeId2;

HLenum nTouchableFace;

//const double kPI = 3.141593;
//const double kFovY = 40;
bool paintCanvas = false;
bool startRecording = false;
bool paintPottery = false;

bool artistButton = true;
bool studentButton = false;
//bool bRenderForce = false;

static HDdouble gSpringStiffness = 0.5;
static HDdouble gMaxStiffness = 1.0;
static HDdouble gStiffnessIncrement = 0.025;

="/******************************************************************/"
/*******************************************************************************

Windows Headers Include files
*******************************************************************************

#include <conio.h>
#include <stdio.h>
#include <assert.h>
#include <process.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sys/types.h>
#include <string>
#include <ctime>
#include <time.h>
#include <corona.h>

#ifdef WIN32
#include <windows.h>
#endif

/*******************************************************************************/

End
*******************************************************************************
/***** constant header file *****/
#define NUMBEROFVOXELS 15
#define CURSOR_SIZE_PIXELS 20
#define RESOLUTION 32
#define WINDOW_WIDTH 1200
#define WINDOW_HEIGHT 900
#define WINDOW_POSITION_X 10
#define WINDOW_POSITION_Y 10
#define PI 3.14159265
//3.1415926535897932384626433832795
#define POINTSIZE 1.0
#define NORMAL_DATA_DISPLAY 1
#define TRACE_DATA_DISPLAY 2
#define FORCE_DATA_DISPLAY 3
#define ANGLE_DATA_DISPLAY 4
#define VISUAL_DATA_DISPLAY 5
//#define RAND_MAX 5
Case III:

/***** main header file *****/
#include "wnd.h"
#include "openGL.h"
#include "openHaptics.h"
#include "constants.h"
#include "bmpReader.h"

/ Function prototypes. */
void exitHandler(void);
void drawSceneHaptics();
void drawSceneGraphics();
void drawCursor();
void updateWorkspace();
double determineFPS(void);

/****************************
OpenGL Functions
***************************************************************************

void renderScene(void);
void drawVertex(double xPos, double yPos, double zPos);
void drawCone(double xPos, double yPos, double zPos);
void drawCube(double xPos, double yPos, double zPos);
void drawCylinder(double xPos, double yPos, double zPos);
void drawSphere(double xPos, double yPos, double zPos);
void drawCanvas();
void drawCanvasBase();
void drawCanvasFrame();
void draw3DRoom();
void drawCanvasControlPanel();
void drawCanvasStatusPanel();
void drawCanvasSketch();
void updateCanvasSketch();
void captureArtistStrokes();
void captureArtistSketches();
void createDisplayLists();
void createSketchDisplayLists();
float calculateGreyValue(float value);
double calculate_zLoc_for_force(double fValue);
double calculate_artistStrokeCount(double count, double x, double y);
void drawGraphicRoom(double xPos, double yPos, double zPos);
void drawHapticRoom(double xPos, double yPos, double zPos);
void drawVTKRoom(double xPos, double yPos, double zPos);
void drawMesh(double xPos, double yPos, double zPos);
void drawColorCube();
void drawPolygon(int a, int b, int c, int d);

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

void mouseCallback(int button, int state, int x, int y);
void motionCallback(int x, int y);
void passivemotionCallback(int x, int y);
void keyboardCallback(unsigned char key, int x, int y);
void specialKeysCallback (int key, int x, int y);
void init(camera *cam);

void glutDisplay(void);
void overlayDisplay(void);
void glutIdle(void);
void initGL();
void initHL();
void initScene();
void drawSceneHaptics();
void drawSceneGraphics();
void drawCursor();
void drawCursorExtension();
void drawCursor2(double x, double y, double z);
void drawDotPattern(double x, double y, double z);
void drawButton(double xPos, double yPos, double zPos, bool enable, char *str);
void drawHapticButton(double xPos, double yPos, double zPos);
void drawMetrics();
void updateWorkspace();
void writeToArtTrack();
void writeToArtTrack2();
void readFromArtTrack(char *fileName);
void populateAverageArtTrack();
void populateAverageArtTrack2();
void groupArtTrack();
void writeObjectPointsToFile();
void drawObject();
void readVTKfile();
void writeVTKfile();
void displayVTKfile();
void flushPointsToFile();

double updateHeightDueToGravity(double currentHeight);

double setMediumColor(int component, double value);
void resetApplication();

/*******************************************************************************
End Functions
******************************************************************************/

/*******************************************************************************
Haptics Functions
******************************************************************************/
void button1UpCallbackFun(unsigned int ShapeID);
void button2UpCallbackFun(unsigned int ShapeID);
void button1DownCallbackFun(unsigned int ShapeID);
void button2DownCallbackFun(unsigned int ShapeID);
void MotionCallback(unsigned int ShapeID);
void GraphicsCallback(void);
void startHaptics(void *arg);
void startDrawing(void *arg);
void initializeHaptics();
void HLCALLBACK hapticButton1DownCallback(HLenum event, HLuint object, HLenum thread, HLcache *cache, void *userdata);
void HLCALLBACK hapticButton1UpCallback(HLenum event, HLuint object, HLenum thread, HLcache *cache, void *userdata);
void HLCALLBACK hapticButton2DownCallback(HLenum event, HLuint object, HLenum thread, HLcache *cache, void *userdata);
**End Functions**

************************************
OpenGL Headers Include files

*******************************************************************************
OpenGL Headers Include files
*******************************************************************************

#include <GL/glut.h>
#include "constants.h"

*******************************************************************************
End
*******************************************************************************

*******************************************************************************
OpenGL variables
*******************************************************************************
typedef struct cameras {
    float pos[3];
    float lookAt[3];
    float lookUp[3];
    } camera;

struct artistDrawingMaterialAttributes{
    // double
    hardness;
    // double
    softness;
    bool redComponent;
    bool greenComponent;

bool blueComponent;
    } artDrwMtrAtr;

struct artistCanvasBoundingBox_node
{
    double x;
    double y;
    double z;
    int gCanvasBoundingBoxDL;
    artistCanvasBoundingBox_node *nxt;
};
artistCanvasBoundingBox_node *start_artistCanvasBoundingBox_node = NULL;

struct artistCharcoalParticals_node
{
    double x;
    double y;
    double z;
    double opacity;
    artistCharcoalParticals_node *nxt;
};
artistCharcoalParticals_node *start_artistCharcoalParticals_node = NULL;

struct artistStroke_node
{
    double x;
    double y;
    double z;
}
double brush_gAngle_x;
double brush_gAngle_y;
double brush_gAngle_z;

double brushForce;

double brushVelocity_x;
double brushVelocity_y;
double brushVelocity_z;

double canvasTouchLocation_x;
double canvasTouchLocation_y;
double canvasTouchLocation_z;

double redComponent;
double greenComponent;
double blueComponent;
double greyValue;
double opacity;

double radius;

double zDifference;
double brushMode;
double pointSize;
double strokeCount;

artistCharcoalParticals_node *starting_node_of_artistCharcoalParticals;
artistStroke_node *nxt;
};
artistStroke_node *start_artistStroke_node = NULL;

struct artistSketch_node
{
    double x;
    double y;
    double z;
    double brush_gAngle_x;
    double brush_gAngle_y;
    double brush_gAngle_z;
    double brushForce;
    double brushVelocity_x;
    double brushVelocity_y;
    double brushVelocity_z;
    double canvasTouchLocation_x;
    double canvasTouchLocation_y;
    double canvasTouchLocation_z;
    double redComponent;
    double greenComponent;
    double blueComponent;
    double greyValue;
double zDifference;
double brushMode;
double pointSize;
double strokeCount;
artistSketch_node *nxt;
};
artistSketch_node *start_artistSketch_node = NULL;

struct record_artistSketch_node
{
    double x;
    double y;
    double z;

double brush_gAngle_x;
    double brush_gAngle_y;
    double brush_gAngle_z;

double redComponent;
    double greenComponent;
    double blueComponent;
    double greyValue;

double pointSize;
    double count;

    DWORD tickCount;
record_artistSketch_node *nxt;

};
record_artistSketch_node *start_record_artistSketch_node = NULL;

struct artistStroke_displayList_node
{
    int gStrokesDL;
    artistStroke_displayList_node *nxt;
};
artistStroke_displayList_node *start_artistStroke_displayList_node = NULL;

struct artistSketch_displayList_node
{
    int gSketchDL;
    artistSketch_displayList_node *nxt;
};
artistSketch_displayList_node *start_artistSketch_displayList_node = NULL;

struct record_artistStroke_node
{
    double x;
    double y;
    double z;
    double brush_gAngle_x;
    double brush_gAngle_y;

double brush_gAngle_z;

double brushForce;

double brushVelocity_x;
double brushVelocity_y;
double brushVelocity_z;

double canvasTouchLocation_x;
double canvasTouchLocation_y;
double canvasTouchLocation_z;

double greyValue;
double zDifference;
double brushMode;
double pointSize;
double strokeCount;

record_artistStroke_node *nxt;
};
record_artistStroke_node *start_record_artistStroke_node = NULL;

struct node
{
    double x;
    double y;
    double z;
    double greyValue;
    node *nxt;
};

struct ppnode
{
    double x;
    double y;
    double z;
    double greyValue;
    ppnode *nxt;
};
ppnode *pp_start_ptr = NULL;

struct surfacepaintnode
{
    double x;
    double y;
    double z;
    double greyValue;
    surfacepaintnode *nxt;
};
surfacepaintnode *spN_start_ptr = NULL;

struct trianglStripNode
{
    double x;

double y;
double z;
double greyValue;
trianglStripNode *nxt;
};

trianglStripNode *tSN_start_ptr = NULL;

struct quadStripNode
{
    double x;
    double y;
    double z;
    double greyValue;
    quadStripNode *nxt;
};
quadStripNode *qSN_start_ptr = NULL;

struct dsnode
{
    //static GLuint gStrokesDisplayList;
    int gStrokesDL;
    dsnode *nxt;
};

dsnode *ds_start_ptr = NULL;
GLuint gStrokesDisplayList = 0;

struct nodeToFile
{
    double x;
    double y;
    double z;
    double xG;
    double yG;
    double zG;
    double xF;
    double yF;
    double zF;
    double greyValue;
    double stressConstant;

    nodeToFile *nxt;
};

nodeToFile *nTF_start_ptr = NULL;

struct volumeNode
{
    double x;
    double y;
    double z;
    greyValue;
};
volumeNode *nxt;
}

volumeNode *volumeNode_start_ptr = NULL;

//enum{
    //formula1,
    //formula2,
    //formula3,
    //formula4,
    //formula5,
    //formula6,
    //formula7,
    //formula8,
//}TRIGONO;

//TRIGONO trigonoselection = formula1;

int formulaChoice = 1;

static camera cam;

static int mainWindow;
static int border = 6, h=500, w=800;

int cursor_X = 0;
int cursor_Y = 0;

int old_cursor_X = 0;
int old_cursor_Y = 0;
int new_cursor_X = 0;
int new_cursor_Y = 0;

int programMode = 3;

GLfloat vertices[][3] = {{-1.0, -1.0, -1.0},
                         { 1.0, -1.0, -1.0},
                         { 1.0,  1.0, -1.0},
                         {-1.0,  1.0, -1.0},
                         {-1.0,-1.0,  1.0},
                         { 1.0,-1.0,  1.0},
                         { 1.0,  1.0,  1.0},
                         {-1.0,  1.0,  1.0}};

GLfloat normals[][3] = {{-1.0, -1.0,-1.0},
                        { 1.0, -1.0,-1.0},
                        { 1.0,  1.0,-1.0},
                        {-1.0,  1.0,-1.0},
                        {-1.0,-1.0,  1.0},
                        { 1.0,-1.0,  1.0},
                        { 1.0,  1.0,  1.0}};

140
GLfloat colors[][] = {{0.0, 0.0, 0.0},
                    {1.0, 0.0, 0.0},
                    {1.0, 1.0, 0.0},
                    {0.0, 1.0, 0.0},
                    {0.0, 0.0, 1.0},
                    {1.0, 0.0, 1.0},
                    {1.0, 1.0, 1.0},
                    {0.0, 1.0, 1.0}};

GLfloat colors[][] = {{0.7, 0.7, 0.7},
                    {0.7, 0.7, 0.7},
                    {0.7, 0.7, 0.7},
                    {0.7, 0.7, 0.7},
                    {0.7, 0.7, 0.7}};

// GL_LIGHT0: the white light emitting light source
// Create light components for GL_LIGHT0
float ambientLight0[] = { 1.0f, 1.0f, 1.0f, 1.0f };
float diffuseLight0[] = { 0.8f, 0.8f, 0.8f, 1.0f };
float specularLight0[] = { 0.5f, 0.5f, 0.5f, 1.0f };
float position0[] = { -10.0f, 5.0f, -10.0f, 1.0f };

// GL_LIGHT1: the red light emitting light source
// Create light components for GL_LIGHT1
float ambientLight1[] = { 1.0f, 0.0f, 0.0f, 1.0f };
float diffuseLight1[] = { 1.0f, 0.0f, 0.0f, 1.0f };
float specularLight1[] = { 1.0f, 0.0f, 0.0f, 1.0f };
float position1[] = { 10.0f, 5.0f, -10.0f, 1.0f };

// GL_LIGHT1: the red light emitting light source
// Create light components for GL_LIGHT1
float ambientLight2[] = { 0.0f, 0.0f, 1.0f, 1.0f };
float diffuseLight2[] = { 0.0f, 0.0f, 1.0f, 1.0f };
float specularLight2[] = { 0.0f, 0.0f, 1.0f, 1.0f };
float position2[] = { 10.0f, 5.0f, 10.0f, 1.0f };

float xDir = 0.0;
float yDir = 0.0;
float zDir = 0.0;

int transp_color;

static GLuint gVTKDisplayList = 0;
static GLuint gArtStrokesDisplayList = 0;
static GLuint gDotPatternDisplayList = 0;
static GLuint gCursorDisplayList = 0;
static GLuint gCursorRingDisplayList = 0;
static GLuint gCanvasDisplayList = 0;
static GLuint gCanvasFrameDisplayList = 0;
static GLuint gCanvasControlPanelDisplayList = 0;
static GLuint gCanvasStatusPanelDisplayList = 0;
static GLuint gArtistBrushForceVisualizationsDisplayList = 0;

static GLuint gMeshDisplayList = 0;
static GLuint gVolumeDisplayList = 0;

static double gCursorScale;

std::string tracedPoints = "";
std::string objectPoints = "";

char tracePoints[200];

float viewAngle = 0.0;
float tippAngle = 0.0;
float elevationCanvas = 0.0;

float paintR = 0.8;
float paintG = 0.8;
float paintB = 0.8;

int *matrix3D;
int matrixDimX = 0;
int matrixDimY = 0;
int matrixDimZ = 0;
int choice_of_data_display = 1;
double surfaceArea = 0.0;
double zoom = 1.0;

double zoomScene = 0.0;
double lookPosition = 1.0;
double rotationAngle = 0.0;
double sceneRotationAngleX = 0.0;
double sceneRotationAngleY = 0.0;
double sceneRotationAngleZ = 0.0;
double sceneScale = 1.0;
double canvasZ = -42.0;
double rad2deg = PI/180;
double radiusXYZ = 0.5;
double canvasTouchZ = 0.0;
double artistStrokeCount = 0.0;
double artistStrokeBrushSize = 1.0;
double artistCanvas_Z_Position = 0.0;
double artistStrokes_viewRotation = 0.0;

/////

double artistCanvas_roll_angle = 0.0;
double artistCanvas_pitch_angle = 0.0;

double matrix3DRearranged[150][150];

/////

double greyVal = 0.0;
double colorIntensity = 0.5;
double freeDist = 0.5;
double opacityValue = 0.5;
double freeRadius = 0.5;
double vertexCount = 0.0;
double max_box_half_diagonal = 0.0;
double line_width = 0.1;
double record_sketch_count = 0.0;
double output_image_file_number = 0.0;
double output_data_file_number = 0.0;

bool surfaceTriangles = false;
bool surfaceVertex = true;
bool randomVertex = true;
bool surfacePaint = false;
bool objectTouched = false;
bool firstPoint = true;
bool firstCircle = true;
bool canvasPaint = false;
bool artistStrokeTraceLocations = false;
bool artistStrokeBrushLocations = true;
bool artistStrokeForceLocations = false;
bool artistStrokeAngleLocations = false;
bool artistStrokeVisualizations = false;
bool freeForm = false;
bool canvasSketch = false;
bool sceneAnimation = false;
bool canvasBoundingBox = true;
bool room3D_Disable = false;
bool playSketch = false;

/******************************************************************************/
OpenHaptics Headers Include files
******************************************************************************

#include <QHHeadersWin32.h>
#include <QHHeadersGLUT.h>
#include <HD/hd.h>
#include <HDU/hduVector.h>
#include <HDU/hduError.h>
#include <HDU/hduMatrix.h>
#include <HDU/hduQuaternion.h>
#include <HDU/hduGenericMatrix.h>
#include <HDU/hduHapticDevice.h>

#include <HL.hl.h>
#include <HLU/hlu.h>

End
******************************************************************************

Haptics variables
******************************************************************************

hduVector3Dd m_devicePosition;

bool omniButton1Down = false;
bool omniButton1Up   = false;
bool omniButton2Down = false;
bool omniButton2Up = false;

hduVector3Dd proxy;

hduVector3Dd proxyRot;

hduVector3Dd deviceReactionForce;

hduVector3Dd canvasTouchCoordinates;

hduVector3Dd omniStylusPosition;

hduVector3Dd omniStylusPositionOld;

hduVector3Dd omniStylusGimbalAngles;

HDdouble omniForce[3];

double xOmniPosition = 0.0;

double yOmniPosition = 0.0;

double zOmniPosition = 0.0;

double xGimbal = 0.0;

double yGimbal = 0.0;

double zGimbal = 0.0;


double xGimbalinDegrees = 0.0;

double yGimbalinDegrees = 0.0;

double zGimbalinDegrees = 0.0;


int x;

int y;

int z;


/* Haptic device and rendering context handles. */

static HHD ghHD = HD_INVALID_HANDLE;

static HHLRC ghHLRC = 0;
/* Shape id for shape we will render haptically. */
HLuint gCanvasShapeId;
HLuint gCanvasFrameShapeId;
HLuint gCanvasControlPanelShapeId;
HLuint gCanvasStatusPanelShapeId;

HLuint gSphereShapeId2;

//const double kPI = 3.141593;
//const double kFovY = 40;
bool paintCanvas = false;
bool startRecording = false;
bool paintPottery = false;

bool artistButton = true;
bool studentButton = false;
//bool bRenderForce = false;

static HDdouble gSpringStiffness = 0.5;
static HDdouble gMaxStiffness = 1.0;
static HDdouble gStiffnessIncrement = 0.025;

/******************************************************************************/
Windows Headers Include files

#include <conio.h>
#include <stdio.h>
#include <assert.h>
#include <process.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sys/types.h>
#include <string>
#include <ctime>
#include <string.h>
#include <ctime>
#include <time.h>
#include <corona.h>

//#if defined(WIN32)
#include <windows.h>
//#endif

End
/***** constant header file *****/
#define NUMBEROFVOXELS 15
#define CURSOR_SIZE_PIXELS 20
#define RESOLUTION 32
#define WINDOW_WIDTH 1200
#define WINDOW_HEIGHT 900
#define WINDOWgetPosition_X 10
#define WINDOWPOSITION_Y 10
#define PI 3.14159265
//3.1415926535897932384626433832795
#define POINTSIZE 1.0

#define NORMAL_DATA_DISPLAY 1
#define TRACE_DATA_DISPLAY 2
#define FORCE_DATA_DISPLAY 3
#define ANGLE_DATA_DISPLAY 4
#define VISUAL_DATA_DISPLAY 5

#define RED 1
#define GREEN 2
#define BLUE 3
#define GREY 4
#define FORMULA1 1
#define FORMULA2 2
#define FORMULA3 3
#define FORMULA4 4
#define FORMULA5 5
#define FORMULA6 6
#define FORMULA7 7
#define FORMULA8 8
PROGRAM : Haptic Enabled Multidimensional Canvas
AUTHOR : KALYAN THOKALA
DATE : 12/06/2010

Description:
Case III 3D Haptic Canvas

Headers Include files

#include "main.h"
using namespace std;
double clk = 0.0;

Initializes GLUT for displaying a simple haptic scene.

int main(int argc, char *argv[]){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowPosition(WINDOW_POSITION_X,WINDOW_POSITION_Y);
    glutInitWindowSize(WINDOW_WIDTH,WINDOW_HEIGHT);
    mainWindow = glutCreateWindow("New Media Arts - Case I");
    glutKeyboardFunc(keyboardCallback);
    glutSpecialFunc(specialKeysCallback);
    glutMouseFunc(mouseCallback);
glutMotionFunc(motionCallback);
glutPassiveMotionFunc(passivemotionCallback);
glutDisplayFunc(glutDisplay);
glutReshapeFunc(glutReshape);
glutIdleFunc(glutIdle);
atexit(exitHandler);
initScene();
glutMainLoop();
return 0;
}

/******************************************************************************
GLUT callback for redrawing the view.
*******************************************************************************/
void keyboardCallback (unsigned char key, int x, int y)
{
switch (key)
{
case 'I':

//room3D_Disable = !room3D_Disable;
if(room3D_Disable == true)
room3D_Disable = false;
else
room3D_Disable = true;
gClearColor(1,1,1,1);
//artistStrokeBrushLocations = true;
//artistStrokeTraceLocations = false;
//artistStrokeForceLocations = false;
//artistStrokeAngleLocations = false;
//artistStrokeVisualizations = false;
//choice_of_data_display = NORMAL_DATA_DISPLAY;

}
break;
case '2':
{

if(room3D_Disable == true)
room3D_Disable = false;
else
room3D_Disable = true;

glClearColor(0,0,0,1);

}
break;
case '3':
{

formulaChoice = 3;

}
break;
case '4':
{

formulaChoice = 4;

}
break;
case '5':
{
    formulaChoice = 5;
}
break;
case '6':
{
    formulaChoice = 6;
}
break;
case '7':
{
    formulaChoice = 7;
}
break;
case '8':
{
    formulaChoice = 8;
}
break;
case 'i':
case 'I':
case 't':
{
    playSketch = !playSketch;
    record_sketch_count = 0.0;
}
break:
case 'o':
  case 'O':
  {
    flushPointsToFile();
  }
  break;

  case 'R':
  case 'r':
  {
    if(canvasPaint)
    {
      if(artDrwMtrAtr.redComponent == true)
      {
        artDrwMtrAtr.redComponent = false;
      }
      else
      {
        artDrwMtrAtr.redComponent = true;
      }
    }
  }
  break;

  case 'z':
  case 'Z':
  {  

{ 
    canvasBoundingBox = !canvasBoundingBox;
}

break;
case 'G':
case 'g':
{
    if(canvasPaint)
    {
        if(artDrwMtrAtr.greenComponent == true)
            artDrwMtrAtr.greenComponent = false;
        else
            artDrwMtrAtr.greenComponent = true;
    }
    break;
}
case 'B':
case 'b':
{
    
}
if(canvasPaint)
{

if(artDrwMtrAtr.blueComponent == true)

artDrwMtrAtr.blueComponent = false;

else

artDrwMtrAtr.blueComponent = true;

}

} break;

case 'C':
case 'c':
{

//if(canvasPaint)

//setMediumColor(GREY);

} break;
case 'F':
    case 'f':
    {
        if(canvasPaint)
        {
            if(freeForm == true)
                freeForm = false;
            else
                freeForm = true;

        }
    }
    break;

    case 'P':
    case 'p':
    {
        //if(canvasPaint)
    }
    break;
if(canvasSketch == true)
    canvasSketch = false;
else
    canvasSketch = true;

break;

if(sceneAnimation == true)
    sceneAnimation = false;
else
    sceneAnimation = true;

case 'A':
    case 'a':
        if(sceneAnimation == true)
            sceneAnimation = false;
else
    sceneAnimation = true;
break;

case 'V':
    case 'v':
    {

        //if(sceneAnimation == true)

        sceneAnimation = false;

        //

        //else

        sceneAnimation = true;

        //

        readVTKfile();

    }

    break;

    case 'w':
        case 'W':
        {

            glReadBuffer(GL_BACK);

        }

        break;


GLvoid *imageData = malloc(WINDOW_WIDTH*WINDOW_HEIGHT*(8+8+8+8));

glReadPixels(0, 0, WINDOW_WIDTH, WINDOW_HEIGHT, GL_RGBA, GL_UNSIGNED_BYTE, imageData);

corona::Image *myImage = corona::CreateImage(WINDOW_WIDTH, WINDOW_HEIGHT, corona::PF_R8G8B8A8, imageData);

corona::FlipImage(myImage, corona::CA_X);

char *fName = new char[30];

sprintf(fName,"imageOutput\%g.png",output_image_file_number);
//fName = "programOutput\scene1.png";
//fName = "imageOutput\scene1.png";
//sprintf(fName,"programOutput\scene1.png",imageNumber);

corona::SaveImage(fName, corona::FF_PNG, myImage);

delete myImage;

free(imageData);

//imageNumber = imageNumber + 1;

output_image_file_number = output_image_file_number + 1;

}

break;

default:

{

choice_of_data_display = NORMAL_DATA_DISPLAY;

}

break;

}
/*******************************************************************************
GLUT callback for redrawing the view.
*******************************************************************************

void motionCallback(int x, int y)
{
    if(programMode == 3 )
    {
        x = x - (WINDOW_WIDTH/2);
        y = y - (WINDOW_HEIGHT/2);
        X = x;
        Y = y;
        double dist_xy = sqrt((x*x*1.0)+(y*y*1.0));
        if(new_cursor_X > old_cursor_X)
        {
            sceneRotationAngleX += dist_xy/100;//0.1;
        }
        else
        {
            sceneRotationAngleX -= dist_xy/100;//0.1;
        }
    }
}

*/
if(new_cursor_Y > old_cursor_Y)
{
    sceneRotationAngleY += dist_xy/100;//0.1;
}
else
{
    sceneRotationAngleY -= dist_xy/100;//0.1;
}
old_cursor_X = new_cursor_X;
old_cursor_Y = new_cursor_Y;

/*******************************************************************************
GLUT callback for redrawing the view.
******************************************************************************/
void mouseCallback(int button, int state, int x, int y)
{
    //printf("button: %d\n", button);
    if(state == GLUT_UP)
    {
    }
switch(button
)
{
    case GLUT_LEFT_BUTTON:
        //sceneScale += 0.01;
        break;
    case GLUT_RIGHT_BUTTON:
        //sceneScale -= 0.01;
        break;
    case GLUT_MIDDLE_BUTTON:
        //printf("GLUT_MIDDLE_BUTTON_UP\n");
        break;
    case GLUT_WHEEL_UP:
        sceneScale += 0.01;
        break;
}
case GLUT_WHEEL_DOWN:
    sceneScale - 0.01;
    break;

default:
    //printf("GLUTn");
    break;
}
}

if(state == GLUT_DOWN)
{
    switch(button )
    {
    case GLUT_LEFT_BUTTON:
        //printf("GLUT_LEFT BUTTON_DOWNn");
        break;
    }
GLUT_RIGHT_BUTTON:

//printf("GLUT_RIGHT_BUTTON_DOWN\n");

break;

GLUT_MIDDLE_BUTTON:

//printf("GLUT_MIDDLE_BUTTON_DOWN\n");

break;

default:

//printf("GLUT\n");

break;

double timeClock=0;

GLUT callback for redrawing the view.
void glutDisplay()
{
    drawSceneHaptics();
    drawSceneGraphics();
    glutSwapBuffers();

    char title[150];
    sprintf(title, "New Media Arts %4.1f fps , Vertex Count : %g", determineFPS(), vertexCount);
    glutSetWindowTitle(title);

    if(sceneAnimation == true)
    {
        int r = rand();
        //if(r%2 == 0)
        sceneRotationAngleX += 0.5;
        //else
        sceneRotationAngleY += 0.5;
    }
}
/***********************************************
Sets up general OpenGL rendering properties: lights, depth buffering, etc.
***********************************************

void initGL()
{

    static const GLfloat light0_ambient[] = {0.3f, 0.3f, 0.3f, 0.3f};
    static const GLfloat light0_diffuse[] = {0.8f, 0.8f, 0.8f, 0.8f};
    static const GLfloat light0_specular[] = {1.0f, 1.0f, 1.0f, 0.6f};
    static const GLfloat light0_direction[] = {0.0f, 0.0f, 5.0f, 1.0f};

    glDepthFunc(GL_LEQUAL);
    glEnable(GL_DEPTH_TEST);

    back faces.

    glCullFace(GL_BACK);
    glEnable(GL_CULL_FACE);

    other misc features.

    glEnable(GL_LIGHTING);
glEnable(GL_NORMALIZE);

glShadeMode(GL_SMOOTH);

// Setup lighting model.

glLightfv(GL_LIGHT0, GL_AMBIENT, light0_ambient);

glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_diffuse);

glLightfv(GL_LIGHT0, GL_SPECULAR, light0_specular);

glLightfv(GL_LIGHT0, GL_POSITION, light0_direction);

glEnable(GL_LIGHT0);

// Blend

glEnable(GL_BLEND);

glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

glEnable(GL_COLOR_MATERIAL);
```cpp
//glColorMaterial

float specularReflection[] = { 0.5f, 0.5f, 0.5f, 0.3f };

//glColorMaterial

glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, specularReflection);

//glColorMaterial

glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, 96);

void drawSceneGraphics()
{

    //glClear

    COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //if((!artistStrokeLocations) && (!artistStrokeVisualizations))

    //if(!room3D_Disable)

    draw3DRoom();

    /***********************************************/

    The main routine for displaying the scene. Gets the latest snapshot of state
    from the haptic thread and uses it to display a 3D cursor.

    /***********************************************/
```
if(canvasBoundingBox)
{
    drawCanvasBase();
    drawMetrics();
}

drawCanvas();

//drawCanvasFrame();
//drawCanvasControlPanel();
//drawCanvasStatusPanel();

//drawCanvasSketch();

//drawMetrics();
drawCursor();

if(!canvasSketch)
    drawCursorExtension();

//if(canvasSketch)
    drawCanvasSketch();

//displayVTKfile();

172
The main routine for rendering scene haptics.
*******************************************************************************
void drawSceneHaptics()
{

// Start haptic frame. (Must do this before rendering any haptic shapes.)
    hlBeginFrame();

    // Start a new haptic shape. Use the feedback buffer to capture OpenGL
    // geometry for haptic rendering.

    hlTouchModel(HL_CONSTRAINT);
    hlTouchModelf(HL_SNAP_DISTANCE, 2.5);

    /*
    if(canvasSket
    ch)
    {
        hlMaterialf(HL_FRONT, HL_STIFFNESS, 0.7f);
        hlMaterialf(HL_FRONT, HL_DAMPING, 0.1f);
        hlMaterialf(HL_FRONT, HL_STATIC_FRICTION, 0.2f);
        hlMaterialf(HL_FRONT, HL_DYNAMIC_FRICTION, 0.3f);

        (HL_SHAPE_FEEDBACK_BUFFER, gCanvasShapeId);
    }

    hlBeginShape
drawCanvas();
hlEndShape();
}
/*
*/

hlMaterialf(HL_FRONT, HL_STIFFNESS, 0.7f);
hlMaterialf(HL_FRONT, HL_DAMPING, 0.1f);
hlMaterialf(HL_FRONT, HL_STATIC_FRICTION, 0.2f);
hlMaterialf(HL_FRONT, HL_DYNAMIC_FRICTION, 0.3f);
hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, gCanvasFrameShapeId);
// Use OpenGL commands to create geometry.
drawCanvasFrame();
// End the shape.
.hlEndShape();

hlMaterialf(HL_FRONT, HL_STIFFNESS, 0.7f);
hlMaterialf(HL_FRONT, HL_DAMPING, 0.1f);
.hlMaterialf(HL_FRONT, HL_STATIC_FRICTION, 0.2f);
.hlMaterialf(HL_FRONT, HL_DYNAMIC_FRICTION, 0.3f);
.hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, gCanvasControlPanelShapeId);
// Use OpenGL commands to create geometry.
drawCanvasControlPanel();
// End the shape.
.hlEndShape();

hlMaterialf(HL_FRONT, HL_STIFFNESS, 0.7f);
.hlMaterialf(HL_FRONT, HL_DAMPING, 0.1f);
hlMaterialf(HL_FRONT, HL_STATIC_FRICTION, 0.2f);

hlMaterialf(HL_FRONT, HL_DYNAMIC_FRICTION, 0.3f);

hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, gCanvasStatusPanelShapeId);
// Use OpenGL commands to create geometry.

tatusPanel();

// End the shape.

hlEndShape();

*/

// Check for events.

hlCheckEvents();

// End the haptic frame.

hlEndFrame();

}
/*******************************************************************************
Draws a 2D canvas.
*******************************************************************************

void draw3DRoom()
{
  glPushMatrix();
  glPushMatrix();
  glScalef(1.35, 1.0, 1.0);
  glPushMatrix();
  glTranslatef(0.0, 0.0, 0.0);
  glBegin(GL_POLYGON);
  glColor3f(1.0, 1.0, 1.0);
  glNormal3f(1.0, 1.0, 1.0);
  glVertex3f(1.0, 1.0, 1.0);
  glColor3f(1.0, 1.0, 1.0);
  glNormal3f(1.0, -1.0, 1.0);
  glVertex3f(1.0, -1.0, 1.0);
  glColor3f(1.0, 1.0, 1.0);
  glNormal3f(1.0, -1.0, -1.0);
  glVertex3f(1.0, -1.0, -1.0);
  glColor3f(1.0, 1.0, 1.0);
  glNormal3f(1.0, -1.0, -1.0);
  glVertex3f(1.0, -1.0, -1.0);
  glEnd();
  glBegin(GL_POLYGON);
  glColor3f(1.0, 1.0, 1.0);
  glNormal3f(-1.0, 1.0, 1.0);
  glVertex3f(-1.0, 1.0, 1.0);
  glColor3f(1.0, 1.0, 1.0);
  glNormal3f(-1.0, 1.0, -1.0);
  glVertex3f(-1.0, 1.0, -1.0);
  glColor3f(1.0, 1.0, 1.0);
  glNormal3f(-1.0, 1.0, -1.0);
  glVertex3f(-1.0, 1.0, -1.0);
  glEnd();
}
glVertex3f(-1.0, -1.0, 1.0);
glColor3f(1.0, 1.0, 1.0);
glNormal3f(-1.0, -1.0, -1.0);
glVertex3f(-1.0, -1.0, -1.0);
glColor3f(1.0, 1.0, 1.0);
glNormal3f(-1.0, 1.0, -1.0);
glVertex3f(-1.0, 1.0, -1.0);
glEnd();

glBegin(GL_POLYGON);

glColor3f(1.0, 1.0, 1.0);
glNormal3f(-1.0, 1.0, 1.0);
glVertex3f(-1.0, 1.0, 1.0);
glColor3f(1.0, 1.0, 1.0);
glNormal3f(-1.0, -1.0, 1.0);
glVertex3f(-1.0, -1.0, 1.0);
glColor3f(1.0, 1.0, 1.0);
glNormal3f(1.0, -1.0, 1.0);
glVertex3f(1.0, -1.0, 1.0);

glEnd();


glBegin(GL_POLYGON);

glColor3f(1.0, 1.0, 1.0);
glNormal3f(-1.0,1.0,-1.0);
glVertex3f(-1.0,1.0,-1.0);
glColor3f(1.0,1.0,1.0);
glNormal3f(-1.0,-1.0,-1.0);
glVertex3f(-1.0,-1.0,-1.0);
glColor3f(1.0,1.0,-1.0);
glNormal3f(1.0,-1.0,-1.0);
glVertex3f(1.0,-1.0,-1.0);

1.0,1.0,1.0);
1.0,1.0,1.0);

1.0,1.0,1.0);
1.0,1.0,1.0);

1.0,-1.0,1.0);
1.0,-1.0,1.0);

1.0,-1.0,1.0);
1.0,-1.0,1.0);

1.0,1.0,-1.0);
1.0,1.0,-1.0);

1.0,1.0,-1.0);
1.0,1.0,-1.0);

1.0,1.0,1.0);
1.0,1.0,1.0);

1.0,1.0,1.0);
1.0,1.0,1.0);

1.0,1.0,1.0);
1.0,1.0,1.0);

1.0,-1.0,-1.0);
1.0,-1.0,-1.0);

1.0,-1.0,-1.0);
1.0,-1.0,-1.0);

1.0,1.0,-1.0);
1.0,1.0,-1.0);

1.0,1.0,-1.0);
1.0,1.0,-1.0);

1.0,-1.0,-1.0);
1.0,-1.0,-1.0);

1.0,-1.0,-1.0);
1.0,-1.0,-1.0);

1.0,-1.0,-1.0);
1.0,-1.0,-1.0);

1.0,1.0,-1.0);
1.0,1.0,-1.0);

1.0,1.0,-1.0);
1.0,1.0,-1.0);

1.0,-1.0,-1.0);
1.0,-1.0,-1.0);

1.0,-1.0,-1.0);
1.0,-1.0,-1.0);
glVertex3f(-1.0,1.0,1.0);
glColor3f(1.0,1.0,1.0);
glNormal3f(1.0,1.0,1.0);
glVertex3f(1.0,1.0,1.0);
glColor3f(1.0,1.0,1.0);
glNormal3f(1.0,1.0,-1.0);
glVertex3f(1.0,1.0,-1.0);
glColor3f(1.0,1.0,1.0);
1.0,1.0,1.0);
1.0,1.0,-1.0);
1.0,1.0,-1.0);
1.0,1.0,-1.0);
1.0,-1.0,1.0);
1.0,-1.0,1.0);
1.0,-1.0,1.0);

179
glColor3f(1.0, 1.0, 1.0);

glNormal3f(1.0, 0.0, 1.0);

glVertex3f(1.0, 0.0, 1.0);

glColor3f(1.0, 0.0, 1.0);

if (canvasBoundingBox)
{

// glutSolidTeapot(0.25);

}
void drawCanvasBase()
{

    /*
     * if (!gCanvasDisplayList)
     *
     */
    if (!gCanvasDisplayList)
    {
        gCanvasDisplayList = glGenLists(1);
        glNewList(gCanvasDisplayList, GL_COMPILE);
        glPushMatrix();
        glLineWidth(0.5);
        glColor3f(1.0, 1.0, 1.0);
    }
    glLineWidth(0.1);
    glTranslatef(0.0, 0.0, 0.0);
    glScalef(0.2, 0.2, 0.2);
    glColor3f(1.0, 0.7, 0.7);
    glutWireCube(1.0);
    glPopMatrix();
}

glPopMatrix();
glTranslatef(0.0, 0.0, 0.0);

glutWireCube(3.0);

glutSolidSphere(0.025, RESOLUTION, RESOLUTION);

//glPushMatrix();

//glTranslatef(0.0, -1.5, 0.0);

//glScalef(3.0, 0.01, 3.0);

//glutSolidCube(1.0);

  // glPopMatrix();

glPopMatrix();

  glEndList();

  }

  */

();

}

*/

neRotationAngleY, 0.0, 1.0, 0.0, 0.0);

neRotationAngleX, 1.0, 0.0, 0.0, 0.0);

  glEndList();

  glPushMatrix();

  glEndList();

  */

();


boundingBox_node_temp = new artistCanvasBoundingBox_node;

boundingBox_node_temp = start_artistCanvasBoundingBox_node;

();

do
{
if (artistCanvasBoundingBox_node_temp == NULL)
{
}
else
{


while (artistCanvasBoundingBox_node_temp != NULL);

); glPopMatrix();

); glPopMatrix();

); glPopMatrix();

if (canvasPaint == true)
{
    double box_half_diagonal = sqrt(proxy[0]*proxy[0]+proxy[1]*proxy[1]+proxy[2]*proxy[2]);
    if (box_half_diagonal > max_box_half_diagonal)
    {
        max_box_half_diagonal = box_half_diagonal;
    }

    glLineWidth(0.5);
    (max_box_half_diagonal*2);

    /*
    */
    ()
    (sce,
    RotationAngleY,0.0,1.0,0.0);
void drawCanvas()
{
    glPushMatrix();
    glRotatef(sceneRotationAngleY, 0.0, 1.0, 0.0);
    glRotatef(sceneRotationAngleX, 1.0, 0.0, 0.0);
    glPushMatrix();
    glScalef(sceneScale, sceneScale, sceneScale);
    glCallList(gCanvasDisplayList);
    glPopMatrix();
    glPopMatrix();
}

neRotationAngleX, 1.0, 0.0, 0.0, 0.0);
isplayList_node *artistStroke_displayList_node_temp;

isplayList_node_temp = new artistStroke_displayList_node;

isplayList_node_temp = start_artistStroke_displayList_node;

();

{

(isplayList_node_temp == NULL)

{} else

{

glPushMatrix();

glCallList(artistStroke_displayList_node_temp->gStrokesDL);

glPopMatrix();

artistStroke_displayList_node_temp = artistStroke_displayList_node_temp->nxt;

}
while (artistStroke_displayList_node_temp != NULL);

} else {

//updateCanv

tch)

if (!canvasSketch) {

//drawItems

double oldPoints[3600][3];

double oldPointsFirst[3600][3];
ode *artistStroke_node_temp;

ode_temp = new artistStroke_node;

ode_temp = start_artistStroke_node;

do {

if (artistStroke_node_temp == NULL)
{
}
else
{

//artistStroke_node_temp->z = updateHeightDueToGravity(artistStroke_node_temp->z);

x = artistStroke_node_temp->x;

double

y = artistStroke_node_temp->y;

double

z = artistStroke_node_temp->z;

double

g = artistStroke_node_temp->greyValue;
double pS = artistStroke_node_temp->pointSize;

double rC = artistStroke_node_temp->redComponent;

double gC = artistStroke_node_temp->greenComponent;

double bC = artistStroke_node_temp->blueComponent;

double oV = artistStroke_node_temp->opacity;

//glPushMatrix

//glTranslatef(x,y,z+artistCanvas_Z_Position);

//glColor3f(0.7,0.7,0.7);

//glScalef(1.0,1.0,0.01);

//glutSolidSphere(0.05,RESOLUTION,RESOLUTION);

//glPopMatrix();
glPushMatrix();

//glPointSize(pS);

glBegin(GL_QUAD_STRIP);

//glBegin(GL_POINTS);

//glColor4f(0.5,0.5,0.7,colorIntensity);

gColor4f(rC,gC,bC,oV);

//glNormal3f(x,y,z);

//glVertex3f(x,y,z+artistCanvas_Z_Position);

int countPoints = 0;

double xP;

double yP;

double zP;

bool firstVertex = true;

firstVertexArray[3];
firstOldVertexArray[3];

double

artistCharcoalParticals_node *artistCharcoalParticals_node_temp;

artistCharcoalParticals_node_temp = new artistCharcoalParticals_node;

artistCharcoalParticals_node_temp = artistStroke_node_temp->starting_node_of_artistCharcoalParticals;

do
{

if (artistCharcoalParticals_node_temp == NULL)
{

}

else
{

}

}
{ 

//printf("gt");

//clk++;

//if(artistStroke_node_temp->z > 0.05)

//artistCharcoalParticals_node_temp->z = updateHeightDueToGravity(artistCharcoalParticals_node_temp->z);

artistCharcoalParticals_node_temp->x;  

artistCharcoalParticals_node_temp->y;  

artistCharcoalParticals_node_temp->z;  

//glVertex3f(xP+0.01*sin(clk),yP+0.01*cos(clk),zP);

//printf("%g",xP);
if(firstVertex)
{
  //if(!firstCircle)
  }

  //firstVertexArray[0] = xP;

  //firstVertexArray[1] = yP;

  //firstVertexArray[2] = zP;

  //}

  //if(firstCircle)

  //}

  //oldPointsFirst[0][0] = xP;

  //oldPointsFirst[0][1] = yP;
//oldPointsFirst[0][2] = zP;

//}

if(!firstCircle)
{
    glColor4f(rC,gC,bC,artistCharcoalParticals_node_temp->opacity);
    glNormal3f(oldPoints[countPoints][0],oldPoints[countPoints][1],oldPoints[countPoints][2]);
    glVertex3f(oldPoints[countPoints][0],oldPoints[countPoints][1],oldPoints[countPoints][2]);
    glNormal3f(xP,yP,zP);
    glVertex3f(xP,yP,zP);
// if(firstPointCircle)
{

// oldPointsFirst[0][0] = xP;

// oldPointsFirst[0][1] = xP;

// oldPointsFirst[0][2] = xP;

// firstPointCircle = false;

// }

oldPoints[countPoints][0] = xP;

oldPoints[countPoints][1] = yP;

oldPoints[countPoints][2] = zP;
artistCharcoalParticals_node_temp = artistCharcoalParticals_node_temp->nxt;

countPoints++;

firstVertex = false;

while (artistCharcoalParticals_node_temp != NULL);

//if(!firstCircle)

//glNormal3f(oldPointsFirst[0],oldPointsFirst[1],oldPointsFirst[2]);

//glVertex3f(oldPointsFirst[0],oldPointsFirst[1],oldPointsFirst[2]);

//glNormal3f(firstVertexArray[0],firstVertexArray[1],firstVertexArray[2]);

//glVertex3f(firstVertexArray[0],firstVertexArray[1],firstVertexArray[2]);
firstCircle = false;

glEnd();

glPopMatrix();

artistStroke_node_temp = artistStroke_node_temp->nxt;

while (artistStroke_node_temp != NULL);

}
void drawCanvasFrame()
{
    if (!gCanvasFrameDisplayList)
    {
        gCanvasFrameDisplayList = glGenLists(1);
        glNewList(gCanvasFrameDisplayList, GL_COMPILE);

        glPushMatrix();
        glColor3f(0.1, 0.3, 0.9);
        glPushMatrix();
        glTranslatef(0.0, 1.2, 0.0);
        glPushMatrix();
        glScalef(2.0, 0.1, 0.2);
        glutSolidCube(1.0);
        glPopMatrix();
        glPopMatrix();
        glPushMatrix();
        glTranslatef(0.0, -1.2, 0.0);
        glPushMatrix();
        glScalef(2.0, 0.1, 0.2);
        glutSolidCube(1.0);
        glPopMatrix();
        glPopMatrix();
    }
}

198
glPushMatrix();

glTranslatef(-1.0, 0.0, 0.0);

glPushMatrix();

glScalef(0.1, 2.5, 0.2);

glutSolidCube(1.0);

glPopMatrix();

glPopMatrix();

glPushMatrix();

glTranslatef(1.0, 0.0, 0.0);

glPushMatrix();

glScalef(0.1, 2.5, 0.2);

glutSolidCube(1.0);

glPopMatrix();

glPopMatrix();

}
void drawCanvasControlPanel()
{
    if (!gCanvasControlPanelDisplayList)
    {
        gCanvasControlPanelDisplayList = glGenLists(1);
        glNewList(gCanvasControlPanelDisplayList, GL_COMPILE);
        glPushMatrix();
        glTranslatef(-1.3, 0.0, 0.0);
        glPushMatrix();
        glColor3f(0.5, 0.5, 0.5);
        glScalef(0.75, 2.5, 0.1);
        glutSolidCube(1.0);
        glEndList();
    }
    glCallList(gCanvasControlPanelDisplayList);
}

void drawCanvasStatusPanel()
{
    if (!gCanvasStatusPanelDisplayList)
{  
gCanvasStatusPanelDisplayList = glGenLists(1);  
  glNewList(gCanvasStatusPanelDisplayList, GL_COMPILE);  

  glPushMatrix();  
  glTranslatef(1.3, 0.0, 0.0);  
  glPushMatrix();  
  glColor3f(0.5,0.5,0.5);  
  glScalef(0.75,2.5,0.1);  
  glutSolidCube(1.0);  
  glPopMatrix();  
  glPopMatrix();  
  glEndList();  
}  
  glCallList(gCanvasStatusPanelDisplayList);  
}

double grav = 0.0;

void HLCALLBACK hapticEventMotionAnyObjectCallback(HLenum event, HLuint object, HLenum thread, HLcache *cache, void *userdata)
{
    hlCacheGetDoublev(cache, HL_PROXY_POSITION, proxy);
    hlCacheGetDoublev(cache, HL_PROXY_ROTATION, proxyRot);
    hlCacheGetDoublev(cache, HL_DEVICE_FORCE, deviceReactionForce);
    hdGetDoublev(HD_CURRENT_GIMBAL_ANGLES, omniStylusGimbalAngles);
if (canvasPaint == true)
{
    // printf(" z: 
    %g ",proxy[2]*100000)-5000);

    if (!canvasSketch)
        captureArtistStrokes();
    else
        captureArtistSketches();
}

double dist = proxy.magnitude();
Dd forceVec(0,0,0);

//

// forceVec = -0.1*proxy;
//

= sqrt(x*x+z*z);

// force.set(dist,dist,dist);

hdBeginFrame(hdGetCurrentDevice());
hdSetDouble(HD_CURRENT_FORCE, forceVec);
hdEndFrame(hdGetCurrentDevice());
void HLCALLBACK hapticButton1DownCallback(HLenum event, HLuint object, HLenum thread, HLcache *cache, void *userdata)
{
    canvasPaint = true;
}

void HLCALLBACK hapticButton1UpCallback(HLenum event, HLuint object, HLenum thread, HLcache *cache, void *userdata)
{
    canvasPaint = false;
    createDisplayLists();
    if(canvasSketch)
    {
        createSketchDisplayLists();
    }
}

void HLCALLBACK hapticButton2DownCallback(HLenum event, HLuint object, HLenum thread, HLcache *cache, void *userdata)
{
}

void HLCALLBACK hapticButton2UpCallback(HLenum event, HLuint object, HLenum thread, HLcache *cache, void *userdata)
{
}
if (!canvasPaint)
{

    start_artistStroke_node = NULL;
    node = NULL;
    start_record_artistStroke_node = NULL;
    vertexCount = 0.0;

    sceneScale = 1.0;
    sceneRotationAngleY = 0.0;
    sceneRotationAngleX = 0.0;

    glDeleteLists(gArtistBrushForceVisualizationsDisplayList, 1);
}

void HLCALLBACK hapticTouchAnyObjectCallback(HLenum event, HLuint object, HLenum thread, HLCache *cache, void *userdata)
{
}
if(object == gCanvasShapeId)
{
    hlCacheGetDoublev(cache, HL_PROXY_POSITION, canvasTouchCoordinates);
    canvasTouch Z = proxy[2];
    artistStrokeCount = artistStrokeCount + 1;
}

void HLCALLBACK hapticUnTouchAnyObjectCallback(HLenum event, HLuint object, HLMenum thread, HLCache *cache, void *userdata)
{
    if(canvasPaint && canvasSketch)
    {
        createSketch DisplayLists();
    }
    canvasPaint = false;
}
void drawGraphicRoom(double xPos, double yPos, double zPos)
{
    glPushMatrix();

    glPushMatrix();

    glTranslatef(xPos, yPos, zPos);

    //glScalef(0.5,0.5,0.5);
    //glColor3f(0.0,0.0,1.0);
    //glutWireCube(1.0);
    //glutSolidCube(0.5);

    glBegin(GL_POLYGON);
    glColor3f(1.0,1.0,1.0);
    glNormal3f(1.0,1.0,1.0);
    glVertex3f(1.0,1.0,1.0);
    glColor3f(1.0,1.0,1.0);
    glNormal3f(1.0,-1.0,1.0);
    glVertex3f(1.0,-1.0,1.0);

    glEnd();
}

glColor3f(1.0,1.0,1.0);

glNormal3f(1.0,-1.0,-1.0);

glVertex3f(1.0,-1.0,-1.0);

glColor3f(1.0,1.0,1.0);

glNormal3f(1.0,1.0,-1.0);

glVertex3f(1.0,1.0,-1.0);

glEnd();

glBegin(GL_POLYGON);

1.0,-1.0,-1.0;
1.0,1.0,1.0;
1.0,-1.0,1.0;
1.0,-1.0,1.0;

1.0,1.0,1.0;

1.0,-1.0,1.0;
1.0,-1.0,1.0;
1.0,-1.0,-1.0;
1.0,1.0,-1.0;
1.0,1.0,1.0;

1.0,-1.0,1.0;
1.0,-1.0,1.0;
1.0,-1.0,-1.0;

1.0,1.0,1.0;
1.0,-1.0,1.0;
1.0,-1.0,1.0;

1.0,1.0,1.0;
1.0,-1.0,-1.0;
1.0,-1.0,-1.0;

1.0,1.0,1.0;
1.0,-1.0,1.0;
1.0,-1.0,1.0;

1.0,1.0,1.0;
1.0,-1.0,-1.0);

    glEnd();

    glBegin(GL_POLYGON);

    glColor3f(1.0,1.0,1.0);
    glVertex3f(-1.0,1.0,-1.0);
    glColor3f(1.0,1.0,1.0);
    glVertex3f(-1.0,-1.0,-1.0);
    glColor3f(1.0,1.0,1.0);
    glVertex3f(-1.0,-1.0,1.0);
    glColor3f(1.0,1.0,1.0);
    glVertex3f(1.0,-1.0,1.0);
    glEnd();
glColor3f(1.0,1.0,1.0);

glNormal3f(1.0,1.0,1.0);

glColor3f(1.0,1.0,1.0);

glEnd();

glBegin(GL_POLYGON);

1.0,1.0,-1.0);

1.0,1.0,-1.0);

1.0,1.0,-1.0);

1.0,-1.0,-1.0);

1.0,-1.0,-1.0);

1.0,-1.0,-1.0);

1.0,-1.0,-1.0);

1.0,1.0,-1.0);

1.0,1.0,-1.0);

1.0,-1.0,-1.0);

1.0,-1.0,-1.0);

1.0,-1.0,-1.0);

1.0,1.0,-1.0);

1.0,1.0,-1.0);

1.0,-1.0,-1.0);

1.0,-1.0,-1.0);

1.0,-1.0,-1.0);

1.0,1.0,-1.0);

1.0,1.0,-1.0);

1.0,-1.0,-1.0);

1.0,-1.0,-1.0);

1.0,1.0,-1.0);

1.0,1.0,-1.0);

1.0,-1.0,-1.0);

1.0,-1.0,-1.0);

1.0,1.0,-1.0);

1.0,1.0,-1.0);

1.0,-1.0,-1.0);

1.0,-1.0,-1.0);

1.0,1.0,-1.0);

1.0,1.0,-1.0);

1.0,-1.0,-1.0);

1.0,-1.0,-1.0);

1.0,1.0,-1.0);

1.0,1.0,-1.0);

1.0,-1.0,-1.0);

1.0,-1.0,-1.0);

1.0,1.0,-1.0);

1.0,1.0,-1.0);

1.0,-1.0,-1.0);

1.0,-1.0,-1.0);

1.0,1.0,-1.0);

1.0,1.0,-1.0);

1.0,-1.0,-1.0);

1.0,-1.0,-1.0);
glNormal3f(1.0,1.0,-1.0);

glVertex3f(1.0,1.0,-1.0);

glBegin(GL_POLYGON);

setColor3f(1.0,1.0,1.0);

glNormal3f(-1.0,1.0,1.0);

 glVertex3f(-1.0,1.0,1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(1.0,1.0,-1.0);

 glVertex3f(1.0,1.0,-1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(-1.0,1.0,-1.0);

 glVertex3f(-1.0,1.0,-1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(-1.0,1.0,1.0);

 glVertex3f(-1.0,1.0,1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(1.0,1.0,-1.0);

 glVertex3f(1.0,1.0,-1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(-1.0,1.0,-1.0);

 glVertex3f(-1.0,1.0,-1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(-1.0,1.0,1.0);

 glVertex3f(-1.0,1.0,1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(1.0,1.0,-1.0);

 glVertex3f(1.0,1.0,-1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(-1.0,1.0,-1.0);

 glVertex3f(-1.0,1.0,-1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(-1.0,1.0,1.0);

 glVertex3f(-1.0,1.0,1.0);

setColor3f(1.0,1.0,1.0);

glBegin(GL_POLYGON);

setColor3f(1.0,1.0,1.0);

glNormal3f(-1.0,1.0,1.0);

 glVertex3f(-1.0,1.0,1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(1.0,1.0,-1.0);

 glVertex3f(1.0,1.0,-1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(-1.0,1.0,-1.0);

 glVertex3f(-1.0,1.0,-1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(-1.0,1.0,1.0);

 glVertex3f(-1.0,1.0,1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(1.0,1.0,-1.0);

 glVertex3f(1.0,1.0,-1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(-1.0,1.0,-1.0);

 glVertex3f(-1.0,1.0,-1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(-1.0,1.0,1.0);

 glVertex3f(-1.0,1.0,1.0);

setColor3f(1.0,1.0,1.0);

glBegin(GL_POLYGON);

setColor3f(1.0,1.0,1.0);

glNormal3f(-1.0,1.0,1.0);

 glVertex3f(-1.0,1.0,1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(1.0,1.0,-1.0);

 glVertex3f(1.0,1.0,-1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(-1.0,1.0,-1.0);

 glVertex3f(-1.0,1.0,-1.0);

setColor3f(1.0,1.0,1.0);

glNormal3f(-1.0,1.0,1.0);

 glVertex3f(-1.0,1.0,1.0);

setColor3f(1.0,1.0,1.0);

```cpp
```
glVertex3f(-1.0,1.0,-1.0);
glEnd();

glBegin(GL_POLYGON);

glColor3f(1.0,1.0,1.0);
glNormal3f(-1.0,-1.0,1.0);
glVertex3f(-1.0,-1.0,1.0);
glColor3f(1.0,1.0,1.0);
glNormal3f(1.0,-1.0,1.0);
glVertex3f(1.0,-1.0,1.0);
glColor3f(1.0,1.0,1.0);
glNormal3f(1.0,-1.0,-1.0);
glVertex3f(1.0,-1.0,-1.0);
glColor3f(1.0,1.0,1.0);

glNormal3f(-1.0,-1.0,-1.0);
glVertex3f(-1.0,-1.0,-1.0);
glColor3f(1.0,1.0,1.0);

glEnd();
//glutSolidTeapot(0.25);

glPopMatrix();

glPushMatrix();

glTranslatef(xPos, yPos, zPos);

setScale(0.2, 0.2, 0.2);

//glColor3f(0.0, 0.0, 1.0);

setColor3f(1.0, 1.0, 1.0);

glutWireCube(1.0);

//glutSolidCube(1.0);

glPopMatrix();

//glutSolidTeapot(0.25);

}
void flushPointsToFile()
{

    std::string header = "# Art DataFile Version 1.0\n";

    header.append("POINT_DATA\n");

    header.append("X Y Z V GA1 GA2 GA3 TCNT\n");

    ketch_node *record_artistSketch_node_temp;
    ketch_node_temp = new record_artistSketch_node;
    ketch_node_temp = start_record_artistSketch_node;

    do
    {
        if (record_artistSketch_node_temp == NULL)
        {
        }
        else
        {
            double x = record_artistSketch_node_temp->x;
            double y = record_artistSketch_node_temp->y;
            double z = record_artistSketch_node_temp->z;
            double v = record_artistSketch_node_temp->greyValue;
            double s = record_artistSketch_node_temp->pointSize;
        }
    }
double a1 = record_artistSketch_node_temp->brush_gAngle_x;

double a2 = record_artistSketch_node_temp->brush_gAngle_y;

double a3 = record_artistSketch_node_temp->brush_gAngle_z;

DWORD tCount = record_artistSketch_node_temp->tickCount;

char *xx = new char[10];

char *yy = new char[10];
    char *zz = new char[10];

char *vv = new char[10];

char *ss = new char[10];

char *aa1 = new char[10];

char *aa2 = new char[10];

char *aa3 = new char[10];

char *a_tCount = new char[32];

sprintf(xx, "%g", x);

sprintf(yy, "%g", y);

sprintf(zz, "%g", z);

sprintf(vv, "%g", v);
sprintf(ss,"%g",s);

sprintf(aa1,"%g",a1);

sprintf(aa2,"%g",a2);

sprintf(aa3,"%g",a3);

sprintf(a_tCount,"%d",tCount);

header.append(xx);

header.append(", ");

header.append(yy);

header.append(", ");

header.append(zz);

header.append(", ");

header.append(vv);

header.append(", ");

header.append(ss);

header.append(", ");

header.append(aa1);

header.append(", ");
header.append(aa2);

header.append(",");

header.append(aa3);

header.append(",");

header.append(a_tCount);

header.append(",");

header.append("n");

record_artistSketch_node_temp = record_artistSketch_node_temp->nxt;

}

}

while (record_artistSketch_node_temp != NULL);

d("END\n");

char *fName = new char[30];

sprintf(fName,"sequencedOutput\%g.txt",output_data_file_number);
fstream file_op("sequencedOutput\1.txt",ios::out);

fstream file_op(fName,ios::out);
der;
    file_op.close();

written...;

output_data_file_number = output_data_file_number + 1;

}