Final Seminar Report

Perceptual Computing

Leap Motion interactions and Collision detection to pick objects virtually

By

Sidharth Sahdev **2011AAPS098H**

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(November, 2014)

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

HYDERABAD CAMPUS

Students: **Sidharth Sahdev, B.E. Electronics & Communications**

Sanyam Garg, B.E. Electronics & Communications

Expert: Dr. Tathagata Ray

Guiding Faculty:

Duration: 1st August 2014 to 27th November 2014

Date of Submission: 28th November, 2014

Title of the Project: Perceptual Computing in 3-D: Leap Motion Interactions

Course Design Oriented project

Key Words: leap motion, gestures.

Project Area : Perceptual computing

AT

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

Table of Contents

ACKNOWLEDGEMENTS

Firstly, I am grateful to **Department of Computer Science, BITS-Pilani Hyderabad Campus** for offering us Design Oriented Project (DOP) as a course in our curriculum.

I would like to express my sincere thanks to Dr. Tathagata Ray, Department Of Computer Science, BITS –Pilani Hyderabad Campus for giving us the opportunity to carry out a project in this esteemed organization.

I would like to extend our gratitude to my friends Ankesh Kumar and Sujeath Pareddy who helped us out in every possible manner and supported me from time to time.

Abstract

We study the new possibilities to gesture interfaces that have emerged with a Leap Motion sensor. The Leap Motion is an innovative, 3D motion capturing device designed especially for hands and fingers tracking with precision up to 0.01mm.

Through this project we explore as many functionalities of the leap motion device. We learn how to interact with objects virtually and attempt to detect collision and modify a 3-D object in OpenGL environment.

INTRODUCTION

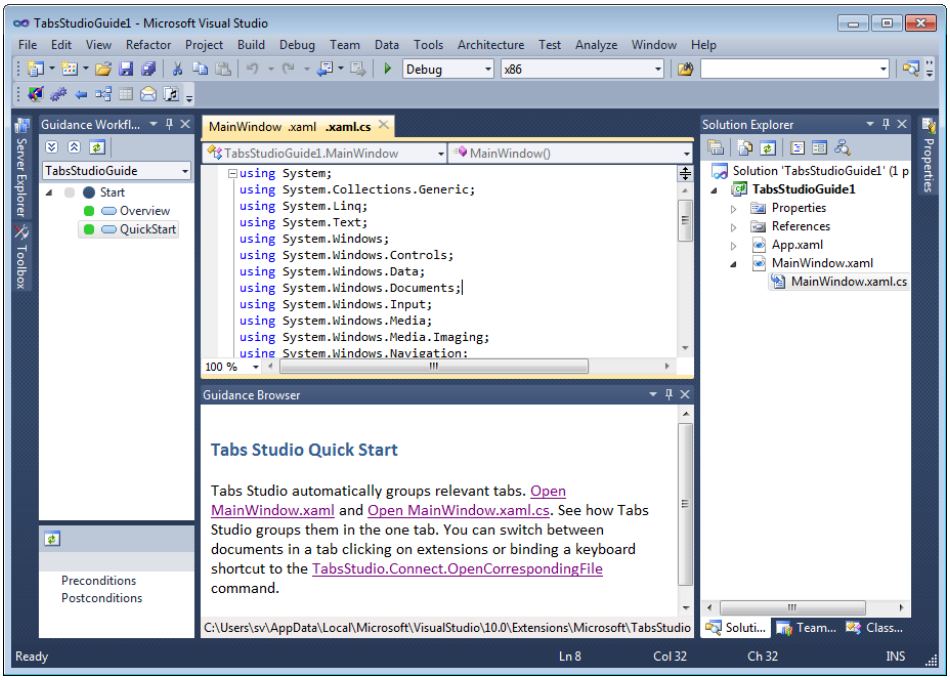
Nowadays, human-computer interaction is mainly based on the pointing or typewriter-style devices. This kind of interaction can limit the natural ways of manipulation using the hands, which may result in a complication of simple tasks. One of the complicated control examples is rotating a three-dimensional object. Using a computer mouse, a user needs to grab the object and rotate it using the mouse, which can only operate in a two-dimensional space. The rotation operation represented by the mouse's movement is unintuitive for humans and users need a few attempts to understand how it works. In the real world, however, the rotation task is natural thus it is simple how to move hands to rotate the object in a desired way.

Therefore, there exists a need for more natural human-computer interfaces. One of the proposed approaches involves hand gestures that are interpreted by a computer. Utilizing hands in a human-computer interface is supported by the fact, that they are even used for non-verbal communication, such as sign or body language. Another advantage of hands is that manipulative tasks performed using hands in real the world could be interpreted as a series of gestures and used as computer input.

The newest sensors provide data that can be successfully used to recognize gestures and therefore control a computer. Currently, there are several devices that yield data useful for the gesture recognition. An example of such a controller is Microsoft Kinect. It provides a three-dimensional point cloud of the observed scene, but was designed for applications that interpret the movement of the whole body of the user. That is why it lacks the needed accuracy for hand gesture recognition.

Another device that is designed to track the movements of a hand and fingers is Leap Motion Controller developed by Leap Motion, Inc. released in July 2013. The Leap Motion is a small device, which can be placed in front of a computer. It features extreme finger detection accuracy up to 0.01 mm. The controller provides information about a position of each finger and a hand detected in the observed space. The SDK attached to the device allows to recognize three predefined gestures: a circle motion with one finger, a swipe action and tapping on the virtual keys. The Leap Motion Controller provides information about every detected hand. This device transmits data with frequency up to 100 Hz. Leap Motion Controller is a sensor that can potentially revolutionize the human-computer interactions.

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop console and graphical applications along with Windows Forms or WPF applications, web sites, web applications, and web services in both native code together with managed code for all platforms supported by Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight.



OpenGL

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three−dimensional applications. Basically OpenGL is nothing more than a set of functions you call from your program (think of it as a collection of .h files). It hides the details of the display adapter, operating system, etc. The API comprises of several libraries with varying levels of abstraction: GL, GLU and GLUT
Levels of abstraction:
GL: lowest level that includes vertex, matrix manipulations. Example: glVertex3f(point.x, point.y, point.z)
GLU: It consists of helper functions for shapes, transformations like gluPerspective(fovy, aspect, near, far)
GLUT: the highest level that deals with window and interface management.
OpenGL is an API, hence we include the header files of it in our program:
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
Windows, linux, UNIX, etc all provide a platform specific implementation. For windows it is opengl32.lib, glu32.lib, glut32.lib. for Linux it is –I GL, -IGLU –I GLUT.

Leap Motion Controller

Leap Motion is an USB sensor device released in July 2013 by Leap Motion Inc., designed to provide real-time tracking of hands and fingers in three-dimensional space with 0:01 millimeter accuracy. It allows a user to get information about objects located in device's field of view (about 150 degree with distance not exceeding 1 meter).

Details of how Leap Motion performs 3D scene capturing have not been revealed by Leap Motion, Inc. However, it is known that hardware consists of three infrared LEDs which are used for scene illumination, while two cameras spaced 4 centimeters apart capture images with 50–200 fps frame rate, dependent whether USB 2.0 or 3.0 is used.

Data Access

Unlike Microsoft Kinect, Leap Motion does not provide access to raw data in the form of a cloud of points. Captured data is processed by proprietary drivers supplied by vendor and accessible through API. Leap Motion was intended to be a human-computer interface, not general purpose 3D scanner, so it is optimized for recognizing human hands and pointy objects.

The main data container we get from Leap Motion API is a Frame. Average frame rate while using dual core laptop and USB 2.0 interface, is 50 frames per second. One frame consists of hands, fingers, pointables (objects directly visible by controller), and additional information, like gestures recognized by simple built-in recognition mechanism, frame timestamp, rotation, translation, and scaling data.

Leap SDK

The Leap device uses a pair of cameras and an infrared pattern projected by LEDs to generate an image of your hands with depth information. A very small amount of processing is done on the device itself, in order to keep the cost of the units low. The images are post-processed on your computer to remove noise, and to construct a model of your hands, fingers, and pointy tools that you are holding.
As an application developer, you can make use of this data via the Leap software developer kit, which contains a powerful high-level API for easily integrating gesture input into your applications. Because developers do not want to go to, the trouble of processing raw input in the form of depth-mapped images skeleton models and point cloud data, the SDK provides abstracted models that report what your user is doing with their hands. With the SDK you can write applications that make use of some familiar concepts:

- All hands detected in a frame, including rotation, position, velocity, and movement since an earlier frame.
- All fingers and pointy tools (collectively known as “pointables”) recognized as attached to each hand, with rotation, position and velocity.
- The exact pixel location on a display pointed at by a finger or tool
- Basic recognition of gestures such as swipe and taps
- Detection of position and orientation changes between frames

Interacting with the Leap Motion.
To interact with the Leap software, we will begin by creating a subclass of Leap::Listener and defining the callback methods we wish to receive. While it is possible to poll the controller for the current frame,

generally you will want to make your program as responsive as possible which is most easily accomplished by acting on input events immediately via callbacks.

INSTALLATIONS

How to Install OpenGL on your System using visual studio:

You need to install:

1. **Visual C++ Express 2010**: Read "How to install Visual C++ Express". VC++ would be installed in "C:\Program Files\Microsoft Visual Studio 10.0\VC", with headers in sub-directory "include" and libraries in "lib".
2. **Windows SDK which includes OpenGL and GLU (OpenGL Utility)**. The Visual C++ 2010 Express bundles the Microsoft Windows SDK, which would be installed in "C:\Program Files\Microsoft SDKs\Windows\v7.0A". (Otherwise, you need to download and install the Windows SDK separately).
   The followings are used from Windows SDK:
   - gl.h, glu.h: header for OpenGL and GLU in directory "C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\gl".
   - opengl32.lib, glu32.lib: libraries for OpenGL and GLU in directory "C:\Program Files\Microsoft SDKs\Windows\v7.0A\lib".
   - opengl32.dll, glu32.dll: dynamic link libraries for OpenGL and GLU in directory "C:\Windows\System32". This directory is to be included in PATH environment variable.

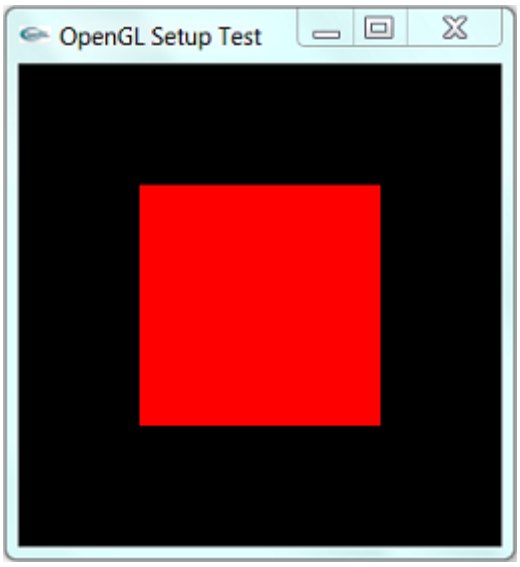
   If you use the VC++ IDE, the include-path and lib-path would have been set correctly. If you use the CMD shell, you need to run the batch file "`vcvarsall.bat`" (in "`C:\Program Files\Microsoft Visual Studio 10.0\VC\bin`"), or "`vcvars32.bat`" in the earlier version, to set the environment variables.
3. **GLUT (OpenGL Utility Toolkit)**: Download Nate Robin's original Win32 port of GLUT from @ http://www.xmission.com/~nate/glut.html (or freeglut @ http://freeglut.sourceforge.net). Unzip and copy "glut.h" to "C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\gl", "glut32.lib" to "C:\Program Files\Microsoft SDKs\Windows\v7.0A\lib", and "glut32.dll" to "C:\Windows\System32" (that is, the same locations as OpenGL and GLU).

Test program: if the following code compiles and works on your computer, you have got OpenGL running on your Laptops/PCs

```
/*
 * GL01Hello.cpp: Test OpenGL C/C++ Setup
 */
#include <windows.h>  // For MS Windows
#include <GL/glut.h>  // GLUT, includes glu.h and gl.h

/* Handler for window-repaint event. Call back when the window first appears and
   whenever the window needs to be re-painted. */
void display() {
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black and opaque
   glClear(GL_COLOR_BUFFER_BIT);         // Clear the color buffer

   // Draw a Red 1x1 Square centered at origin
   glBegin(GL_QUADS);              // Each set of 4 vertices form a quad
      glColor3f(1.0f, 0.0f, 0.0f); // Red
      glVertex2f(-0.5f, -0.5f);    // x, y
      glVertex2f( 0.5f, -0.5f);
```

```
    glVertex2f( 0.5f,  0.5f);
    glVertex2f(-0.5f,  0.5f);
  glEnd();

  glFlush();  // Render now
}

/* Main function: GLUT runs as a console application starting at main()  */
int main(int argc, char** argv) {
  glutInit(&argc, argv);              // Initialize GLUT
  glutCreateWindow("OpenGL Setup Test"); // Create a window with the given title
  glutInitWindowSize(320, 320);   // Set the window's initial width & height
  glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
  glutDisplayFunc(display); // Register display callback handler for window re-paint
  glutMainLoop();          // Enter the infinitely event-processing loop
  return 0;
}
```

You should get this output:



Installing the Leap motion

1. Right click 'My computer' > Properties > Advanced system settings > Enviroment variables > New(under system variables)

Variable name: LEAP_SDK

Variable value: Path to the folder, where you have extracted leap SDK.

1. Since you have already defined LEAP_SDK, all you need to write is '$(LEAP_SDK)\include'.
2. Same as 2. In case you have 64bit machine, you can still go with x86, in case you want to make a software compitable with both - 32bit and 64bit.
3. Yes, only that. You already gave additional folder in previous step, it will look through it and detect Leap.lib.
4. It says to write that, but for some reason it didn't work for me. Just copy required DLLs to folder that contains compiled code. Remember to use Leapd.dll for debug, and Leap.dll for release, just as it said.

5. It's same as 5. Just for release binary. Remember that for release you only need leap.dll, other 2 aren't necessary.

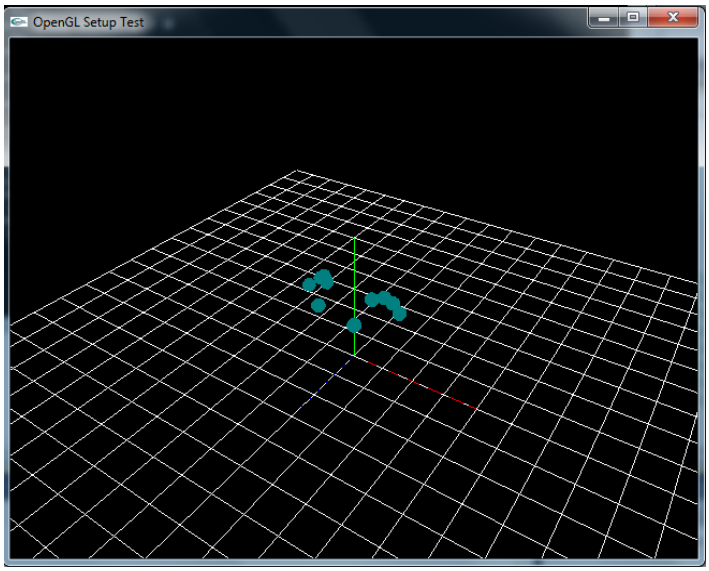
To see if you've done things properly, write a sample code and try to compile it, until debugger doesn't throw errors at you

Exploring the leap motion and openGL

I first tried to implement the basics of OpenGL that included how to draw a square, circle, spheres, cubes, cylinders and polygons.

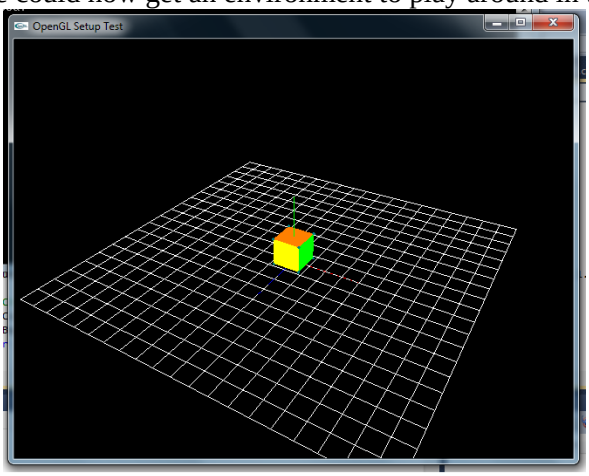
Next task was realizing and defining the proper viewing angle that is needed. I explored the commands like gluortho2d, glulookat, glfrustrum, etc.

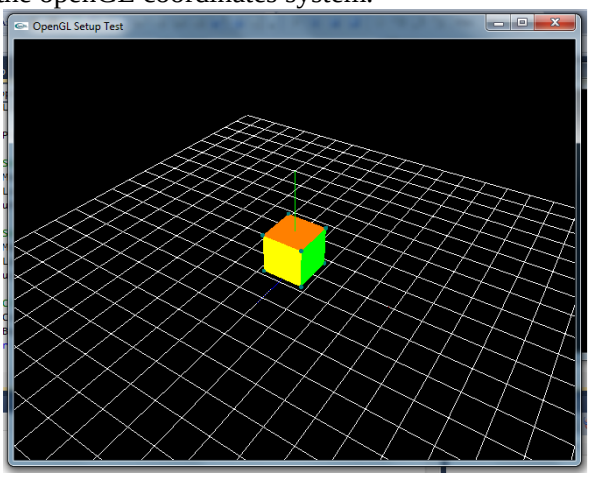
Next I attempted to bring the leap motion data to play with. I plotted the points of the finger tips from the hand on the screen and draw a circle around it to visualize it properly.

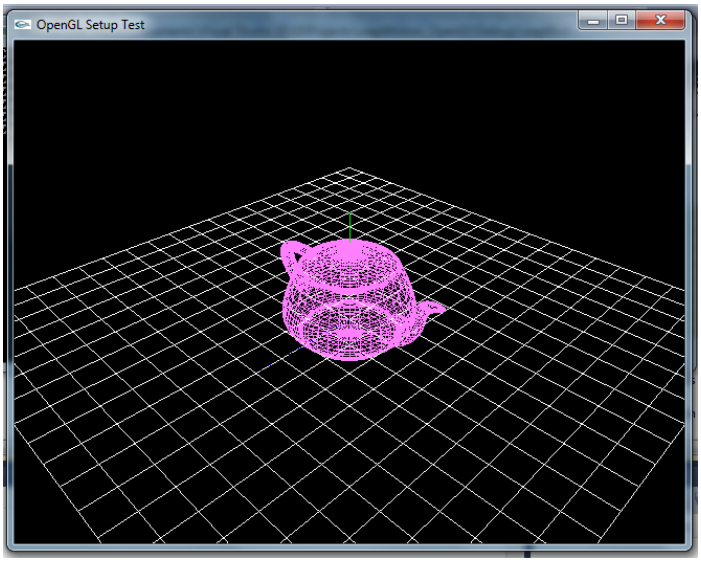


It was noted that the leap motion coordinate system and the OpenGL window coordinate system was not aligned. We therefore made a translation of the leap motion coordinates into the openGL coordinates by dividing it by a factor of 100 and setting the upper and lower bounds.

We could now get an environment to play around in the openGL coordinates system.







Getting the feel of it:

Our next task was to grab the cube and move it around in 3-D space. For doing this we calculated the mean position of 5 fingers. This value, if it is within a certain range around the centroid of the cube, it is assumed to be grabbed and now the cube center is relocated at the mean position of the fingers. A grab function from the leap SDK was also used along with this that takes into account the grab strength. Now if the grab strength is determined by the distance between the fingers… By setting a cutoff for this value the cube can now be accurately grabbed.

Next we provided the ROLL-PITCH-YAW capabilities to the cube using our hands, i.e. after grabbing the cube, the user performs the rotation of his hands in 3-D space and the cube orients itself according to the user's needs.

This was not enough, we had a problem of non-restoring when the hand grab is detected or when it is left and again grabbed. To solve this we subtract the initial roll-pitch-yaw values from the corresponding roll-pitch-yaw when a grab is detected. This ensures that the cube is not suddenly oriented with the hand, but it allows the user to grab the cube from any direction with much confusion!

Interactions:

The first task is to make a 3-D visualizing environment. We make it by drawing the 3 axes (x-y-z) perpendicular to each other on the OpenGL window.

The very first task here is to get all the vertices of the object. We did it for cube by defining the points around the center of the cube. These points were then accommodated for a change in rotation, translation along the x/y/z axes.
Similarly all the eight corners of the cube were calculated and stored in variables. We do this because we will need this information later on.

Collision detection Algorithms:

- ✓ *The problem can be defined as if, where and when two objects intersect.*

- Collision detection, as used in the games community, usually means intersection detection of any form
  - Intersection detection is the general problem: find out if two geometric entities intersect – typically a static problem
  - Interference detection is the term used in the solid modeling literature – typically a static problem
  - Collision detection in the research literature generally refers to a dynamic problem – find out if and when moving objects collide

Choosing an algorithm:

- The geometry of the colliding objects is the primary factor in choosing a collision detection algorithm
  - "Object" could be a point, or line segment
  - Object could be specific shape: a sphere, a triangle, a cube,
  - Objects can be concave/convex, solid/hollow, deformable/rigid, manifold/non-manifold
- The way in which objects move is a second factor
  - Different algorithms for fast or slow moving objects
  - Different algorithms depending on how frequently the object must be updated
- Of course, speed, simplicity, robustness are all other factors

Bounding Volumes:

Reduce complexity of collision computation by substitution of the (complex) original object with a simpler object containing the original one.

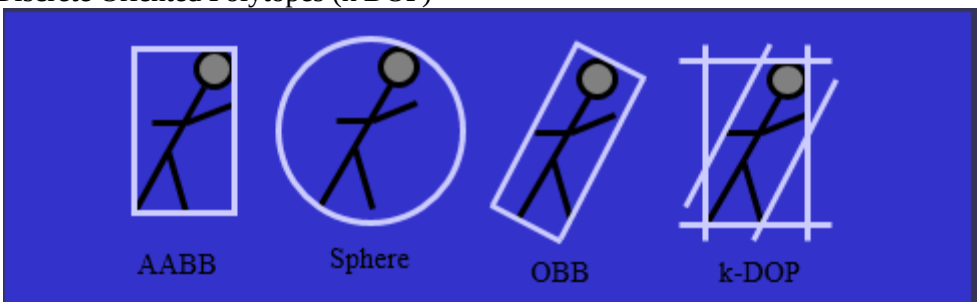
Basic funda: The original objects can only intersect if the simpler ones do. Or better: if the simpler objects do NOT intersect, the original objects won't either.

How to choose Bounding Volumes

- Object approximation behavior ('Fill efficiency')
- Computational simplicity
- Behavior on (nonlinear !) transformation (incl. deformation)
- Memory efficiency

Different BVs:

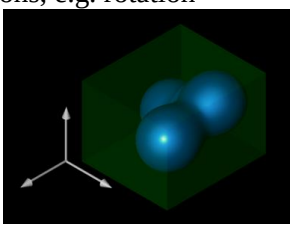
- Axes Aligned Bounding Boxes (AABB)
- Oriented Bounding Boxes (OBB)
- Spheres
- k-Discrete Oriented Polytopes (k DOP)



Overview of the BVs

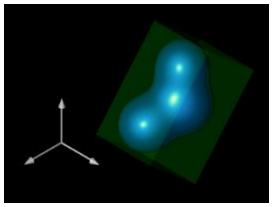
AABBs:

- Align axes to the coordinate system
- Simple to create
- Computationally efficient
- Unsatisfying fill efficiency
- Not invariant to basic transformations, e.g. rotation



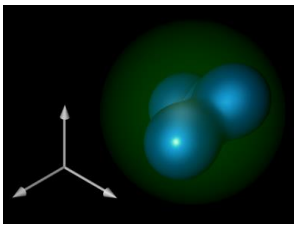
OBB:

Align box to object such that it fits optimally in terms of fill efficiency
Computationally expensive
Invariant to rotation
Complex intersection check



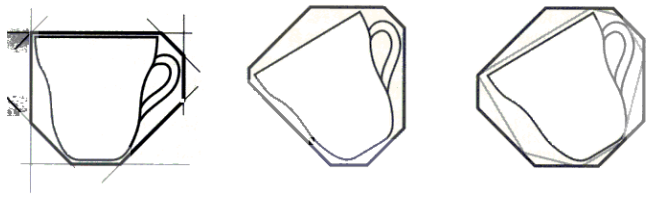
Sphere

- Relatively complex to compute
- Bad fill efficiency
- Simple overlap test
- invariant to rotation



K-DOP

- Easy to compute
- Good fill efficiency
- Simple overlap test
- Not invariant to rotation



k-DOP is considered to be a tradeoff between AABBs and OBBs.
Its collision check is a general version of the AABB collision check, having k/2 directions

Collision test: project BBs onto coordinate axes. If they overlap on each axis, the objects collide.

Using OpenGL

- OpenGL is a rendering system
- OpenGL has no underlying knowledge of the objects it draws
- Collision Detection == Intersection Detection

Collision Detection
- Identifying the intersection of 3D models

Collision Response
- Calculating the appropriate post collision response of the 3D models

Assumptions
- Closed 3D polygonal objects

Overall Approach:

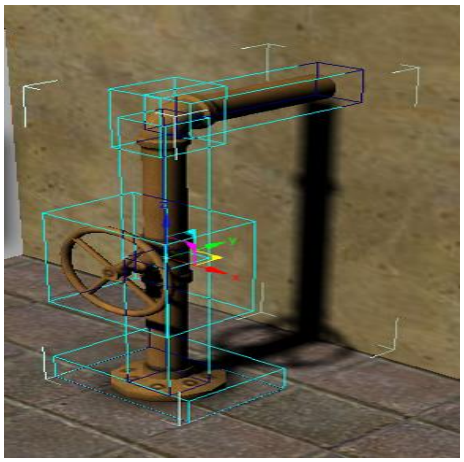
Where is collision detection and response in code?
In animation function

```
Usually in Idle or Display
Update object positions
If (CollisionDetection())
    Collision Response()
Draw Objects
```

Basic Collision Detection function

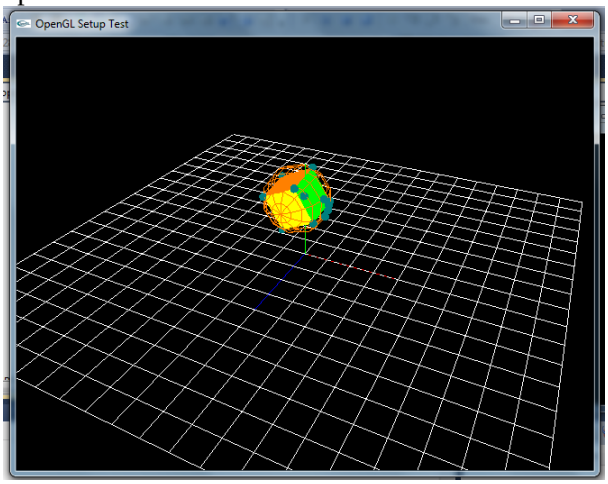
```
for (i=0;i<num_of_objects-1;i++)
    for (j=i+1;j<num_of_objects;j++)
        X=TestIntersection(Object I, Object j)
        If (x==1) return 1;
```

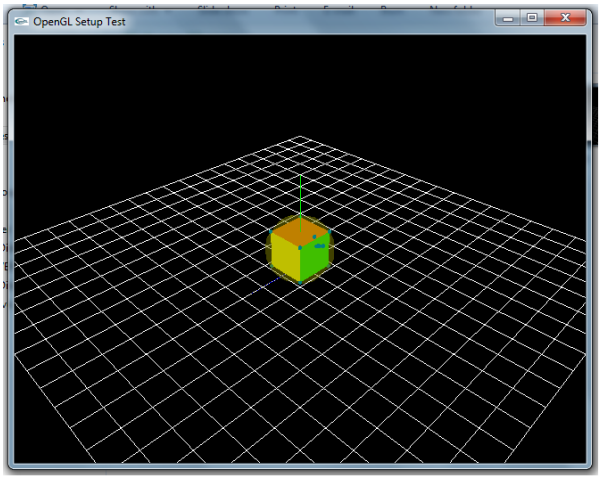
What we want to achieve:
- Calculate a 3D box that bounds the object
- In the collision detection test, use the 3D box to test for collision in place of the full model

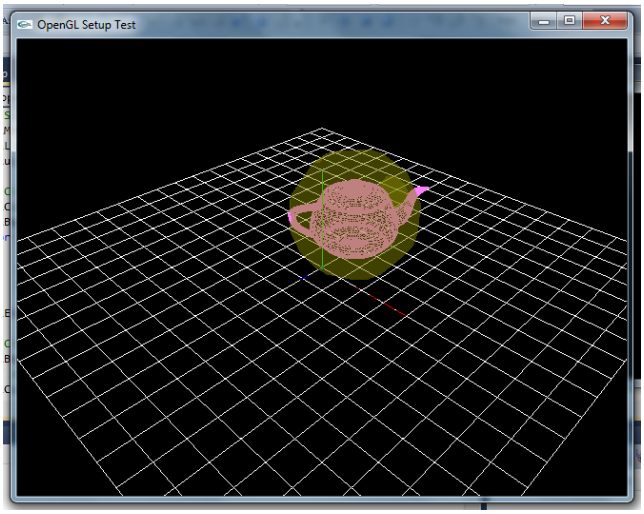


We have taken the “SPHERE” as our Bounding Volume in the project. So when the incoming finger tips enter this sphere we say we have encountered a probable zone of collision.

Implementation:







Colliding plane detection:

First we get the points of the cube accurately under all conditions of roll, pitch, yaw, translate, etc. After we know corners of the cube we can now detect the plane/surface of collision using any of these approaches:

- We define the equation of all the planes made by the 8 points and then decide if the points of the hand lie on any of these plane (from the outside)

OR

- We distribute the 6 faces into set of 4 points
  Find the min & max of the x-y-z coordinates.
  If the hand point lies in the range (min, max) of the set of 4 points, then that particular side/plane of cube is said to have made a collision.

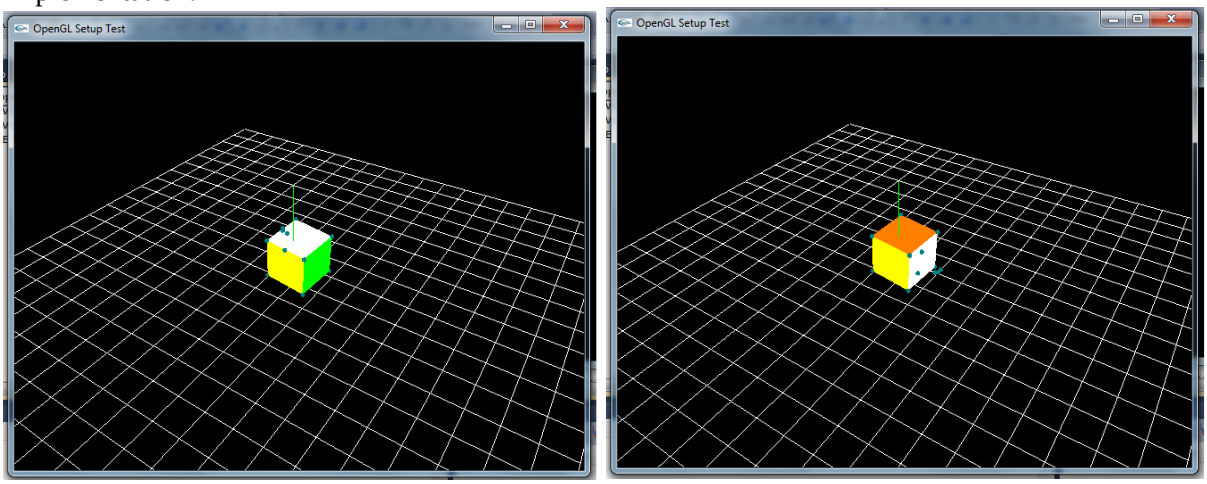
OR

- Another way could be to find the face center of all the 6 planes using the 8 corner points. Compute the distance of the finger from the center points and set a threshold colliding range on distance. This will give you the plane of collision.

OR

- Find the tetrahedron volume enclosed by the faces of the cube and the incoming point. Compute this volume for each of the surface. The volume that has the least value is the most probable surface of collision.

Implementation:



Inclusion Test

We next provide a test for whether the fingertip is inside the cube or outside it. This is primarily done by seeing the sign of the volume of the tetrahedron formed by the fingertip and the surface. If it is +ve, we say that the point is outside the cube. If it is –ve the point is inside the cube.

Modifying the Cube:

Next task was to be able to pinch a vertex of the cube to hold it and place it anywhere in 3-D space. This was achieved by:
Each vertex is given a region of interaction/selection around it in the form of a sphere.
When two points of the finger tips are inside this spherical bounds, we say that the point is up for grab.
The point is relocated with the mean position of the 2 selecting fingers and the polygons corresponding to that vertex are redrawn accordingly.

Future scope & work:

To develop the library for gesture recognition dedicated to Leap Motion device, which will help developers to implement applications using gestures as a human-computer interface.
The goal will be achieved by meeting the following objectives:
_ To design the library architecture,
_ To compare existing methods in the context of hand gesture recognition,
_ To select and implement algorithms for gesture recognition,
_ To implement additional modules enabling the recording and reviewing of gestures saved in a format supported by the library,
_ to create a sample gestures database,
_ to perform gesture recognition tests using Leap Motion Controller.

The additional requirement of the library is that the processing should be in real time!. The recognition modules are based on Support Vector machines (SVM) and Hidden Markov Models (HMM). SVM is a supervised learning algorithm, which uses set of hyper planes for nonlinearly division of input vectors to different classes. HMM is an example of an unsupervised learning algorithm, where model tries to find a hidden structure of data using provided training samples.