

Virtual Boxing(Game Based On Computer Vision)

Development Team: Harshit Maheshwari, Prajya Bhatt, Shubdeep Kochar
Mentor: Ankit Mahato

Description

The purpose of this project is to create a suitable interface between computer graphics and an incoming camera feed which could be usable in other applications in the future. In the present implementation, graphics development has been carried out on OpenGL using GLUT 3.7.6. Computer vision has been handled using OpenCV2.1, with blob detection being achieved using cvblobslib. The game is largely dependent on the quality of the camera feed(at present, the default camera is the only one permitted.) as well as the graphics processing capabilities of the host machine. The game is currently in a very early stage of development, and the major breakthrough of this release is its computer vision aspect.

Technical Manual

The entire development was carried out on Microsoft Visual C++ 2010. The sourcecode consists of the default stafx.h and targetver.cpp and a single file for the project's entire code. The code, in addition requires linking to :

- 1.OpenCV 2.1.0 (<http://opencv.willowgarage.com/wiki/>)
- 2.GLUT 3.7.6 (<http://www.xmission.com/~nate/glut.html>)
- 3.cvblobslib (<http://opencv.willowgarage.com/wiki/cvBlobsLib>)

User-Defined Class Library

The code creates a class library for:

1. Solid Shapes: Cube, Sphere
2. Player: used for storing the basic attributes of a player in the game

All these classes are declared struct and their data members are all public.

The Cube class

Declaration: struct Cube

Data Members (public):

GLfloat edge
GLfloat Centre[3]

Constructors:

- 1.Cube()

The default constructor created by the compiler

- 2.Cube(GLfloat Centre[3], GLfloat edge)

Initialises the cube with the above attributes(edge length and centre's

position in cartesian coordinates)

Member Functions:

void dispCube()

Displays a cube in our openGL environment based on its attributes.

The Sphere class

Declaration: struct Sphere

Data Members(Public):

GLfloat Centre[3];

GLfloat radius;

Constructors:

1.Sphere()

Default constructor created by the compiler

2.Sphere(GLfloat Centre[], GLfloat radius)

Initialises an instance of Sphere with its data members set to the values passed in the arguments

Member Functions:

void dispSphere()

Displays the instance of Sphere with which it is called in our openGL environment.

The Player class

Declaration:

struct Player

Data Members (Public):

Sphere hand1next

Sphere hand2next

Sphere hand1

Sphere hand2

Cube body

GLfloat Centre[3]

GLfloat yspan

GLfloat zspan

GLint orientation

GLint score

Constructor:

Player() : Default constructor

Member Function:

void dispPlayer()

Displays the body data member of the calling object,as well as a temporarily allocated sphere for the head

Global Variables(Classified by type):

IpImage * :hitemp,hitemp1(both grayscale),out, hi,hi2,imgscaling,score,Hi

CvCapture * :hit

int/GLint

maxareahand1, minareahand1, maxareahand2, minareahand2, executeinfinteloop, compspeed, inmoveornot, handedone, handtwodone, timed, m1p1, m1p2, m2p1, m2p2, hh, hl, sl, sh, vh, vl, headcolor, comphandcolor, myhandcolor,count1, count2, decreasestarhand2, decreasestarhand1

float/GLfloat

counthand1, counthand2, speed, xdir, ydir, zdir, xpos, ypos, zpos, tempcomp1[3], tempcomp2[3], q1,q2, x_initpos_hand1, xrandom_hand1, y_initpos_hand1, yrandom_hand1, z_initpos_hand1 ,zrandom_hand1, x_initpos_hand2, xrandom_hand2, y_initpos_hand2, yrandom_hand2, z_initpos_hand2, zrandom_hand2, Zero[3]

char :stringme[22], char stringcomp[29]

Player : me,comp

Standalone Functions:

- GLfloat inline getDistance(GLfloat point1 [],GLfloat point2[])

Accepts two points (cartesian coordinates), returns the Euclidean distance between them

- Overloaded Functions isCollission:

Determine whether an object is colliding with another object

boolean isCollission(Sphere &sphere1,Sphere &sphere2)

Checks whether the arguments passed are colliding

boolean isCollission(Sphere &sphere, Cube &cube)

Checks whether the arguments passed collide

boolean isCollission(Sphere &hand, Player &cur_player,Player &oppn)

The argument passed to hand should belong to the object passed as a reference to the cur_player parameter. The oppn parameter should be passed a reference to the opposing Player. This function checks whether hand collides with oppn.hand, oppn.hand2 or oppn.body.

- The callback functions:

void mouse(int button, int state, int x, int y)

void key(unsigned char key, int x, int y)

Can be used to set responses to key presses and mouse clicks. Their specific implementation is described in the end user's manual.

`void inline reshape (int w,int h)`

Defines the response of glut to a window resize, is passed as functor to `glutReshapeFunc()`

- Functions that are used by the callback functions
Controlling the speed with which camera position is changed(stored in the variable speed)

`void speedincrement ()`
`void speeddecrement()`

The exact effect of pushing a key is defined in the keys function, which calls the following to change camera position and size.

`void xincrement()`
`void yincrement()`
`void zincrement()`
`void xdecrement()`
`void ydecrement()`
`void zdecrement()`
`void xdirdecrement()`
`void ydirdecrement()`
`void zdirdecrement()`
`void nochange()`
`void nowdoasuwish ()`
`void goback()`

- Blob detection and determining the coordinates

`void coordinates(IplImage *out , int check)`

Determines the mean position of the colours green and blue in the image pointed to by out. Blobs corresponding to the hands are filled with these colours in the function `blobbing()` and the image is then sent. These values are set into the respective global variables.

`void blobbing(IplImage *hi, char * win1, char * win2, int check)`

This function contains the blob detection code, which relies on `cvblobslib`. It creates a binary image by checking pixel-by-pixel for the chosen hsv colour range. This image is then passed through the blob detection.

- The Major Functions(Setup of the OpenGL environment):

`void init()`

The call to this function lies in the main loop of GLUT. It contains most of the

settings for the OpenGL environment, such as lighting and shade.

`void display()`

The function passed to the `glutDisplayFunction`. It calls `blobbing` and is also responsible for displaying all the objects in the OpenGL environment. It essentially contains the core of the game's design.

`void todecidescalingfactor()`

It determines the ratio to be used for scaling movement along the direction of view of the webcam. It calls `givedepth()`.

`tobecalledbymain()`

It create the window "score", on which messages as well as It then proceeds to initialise the global `cvCapture` pointer `hit`.

`int main(int argc, char * argv[])`

Initialises many of the global variables. This function calls `todecidescalingfactor()` (which sets framework for depth perception) and then the function `obecalledbymain()`, which contains the main loop for our GLUT.

End User's Manual

Welcome to the world of virtual boxing. In order to enhance your user experience, it is imperative that you go through the following manual.

Getting started:

- Once you launch the application, the program needs to know the position of your hands when you have reached out to your most extreme position and also when your hands are as far back as you would take while normally boxing. This customises the game to your individual style. The game waits for around 10 seconds for your hands to be placed as far out as possible while boxing and then captures the image at that particular instant. This is repeated for your defensive position.
- During this time as well as once the game starts, your hands are shown in a side panel, coloured blue and green. If at any stage one (or both) of your hands fail to be detected, the hands automatically go into a basic defensive position.
- For the best gaming experience, your hands need to be located within the frame of the webcam/ camera that you are using to play the game. You can check the side panel if your hands are being detected properly. (You should see one blue and one green region corresponding to either hand on the panel.)
- During gameplay, your hands appear as spheres on the main window, with the camera position roughly corresponding to your own eyes.
- It is important for the lighting to be sufficiently bright. Again, check the side panel to see if the program is working;
- Note that the program may not work properly if your hands do not stay closed (as gloves do actually constrain them to be in boxing). Gloves of the colour being used will prove ideal. Since colour is used to detect the gloves, it is important for your clothing as well as the background to not have large areas (comparable to your hands in size) of the same colour.

Note: The camera position in the game can be changed. Movement along three perpendicular directions is possible through the following pairs of keys using the keys: 'w' and 's', 'd' and 'a', and 'z' and 'x'.

The camera direction can be altered through rotation about three mutually perpendicular axes using the following pairs of keys: 'j' and 'l', 'o' and 'p', and 'k' and 'i'.