Development of Computer Generated Human Hand

M. Hull

June 11, 2003

Abstract

This Project involves the creation of a user-controlled, graphically-animated human hand for possible use in the future as an interactive tool.

1 Introduction

The program description is quite self-explanatory. The program in OpenGL will be a 3 dimensional graphical animation of a human hand. The thumb and four fingers will have 3 digits, each with its own shortcut key for moving fingers. Unfortunately, access to some type of hardware (ex. a glove) for inputting the position of the fingers is difficult to find. Without it, slow keyboard input is the only capability for motion of the hand and fingers.
2 Background

As computers have become more advanced, more people use them. As technology became cheaper and more useful, many people began to use computers with little to no outside experience with them. Few users know exactly how the interface functions, but they understand how to use it. One of the components that made computers easier to understand was the mouse. Imagine using your computer without the use of the mouse. It led to more complex graphical environments, allowing the use of different windows opened simultaneously. The mouse converted the interface from a one-dimensional text-base to a two-dimensional, slightly graphical, rectangular window-base as computers gained such capabilities. The purpose of the project is to allow the interface to become three dimensional through the use of a graphically animated hand. This third dimension will allow for infinitely more capabilities of the personal computer.

3 Workplan

The goal is to create a fully functional working hand. This means that the model will not only have to look like a hand, but be able to bend and contort exactly as a human hand would be able to.

3.1 model

A main goal of the project is to make the model as realistic as possible. The palm will be made of a torus with planes between them. This will give the palm thickness around the
outside and slightly thinner through the middle. This is much more realistic than a flattened sphere. The fingers will appear best as cylinders and spheres. Each joint will be a sphere so, when the finger is bent, the sphere rounds out the joint. Cylinders will also appear best for the actual hand. The thumb will have similar structure, but many more capabilities, similar to a human thumb. Also, some different additions will be required for aesthetic appeal.

3.2 motion

The model will need to be able to handle any type of motion and position that a real hand would be able to control. In order for each joint to be able to turn to every available position, there will have to be rather tiny increments of motion. I plan to make each joint controlled with a different key press, due to lack of some type of glove. Because of the small increments of motion, I will have an overall modification key set. This set will be able to change the joint angles in much larger changes at a time. This will make it easiest for the hand to make it to any position available.

4 Results

The result of my project is the functional model. I reached some difficulties with aesthetic appeal. The knuckles protrude from the side of the hand extraneously, so I added cylinders running from them to the palm to make the hand appear realistic. The fingers appeared strange when the radius of the joints were larger than the radius of the cylinders of the digits, so I made the two equal to successful results. The fingers also needed to have similar
proportions to a real hand. I set up a function to store length and radius of each digit in arrays. The middle finger is the biggest on the human hand, so the joint at the base of the middle finger is the largest and the longest on my model. I have completed the large scale angle changing function using the number keys, with '0' making a flat hand and '9' making a fist. The specific motion is still under construction. Also, the thumb has been added, but need to be improved for realistic appearance.

5 Code

Once the program was completed, the code was reduced and compacted. What had been nearly a thousand lines was reduced to around 400 lines. This makes the code much more organized. For example, instead of creating a large graphical function for each individual finger, each with its own data and variables, a general function was made that takes some parameters and draws the finger. By this method, I was able to make my code easier to read and understand, but this also significantly shortened the code length itself.

5.1 global variables and constants

My program required a lot of data to be stored globally in order to save having to send it all as parameters back and forth between each individual function. The comments generally define what the different data structures are used for in the program. The exact specifics of how they are used will be described as each on reappears in the code.

#include <stdlib.h>
#include <GL/glut.h>
#include <iostream.h>
#include <time.h>

double pi = 3.1415926535898;

double winsize(600); // size of square window

int xo=0, yo=0; // variables for

int xrot=0, yrot=0; // rotation of screen

int F=20; // number of frames for spheres and cylinders

bool cyl[] = {true, false, false, true}; // decides whether or not a cylinder

// is drawn connecting the knuckle and the palm

int spread[4]; // angle of spread between the different fingers

double joint[5][4]; // sizes of the joints of each finger

double dist[5][3]; // length of each digit

int angle[5][3]; // angle at each joint

GLUquadricObj *qobj;

5.2 main

This is just a basic main function for OpenGL. The first few lines simply enact the use of OpenGL. Comments generally describe what happens with each line. The calls to glut_...
int main(int argc, char **argv)
{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

glutInitWindowSize(int(winsize), int(winsize)); // at this size

glutCreateWindow("Hand"); // actually create it with this title

//srand(time(0)); // randomization: currently unnecessary

myinit(); // my initialization

glutReshapeFunc(myReshape); // reshape func

glutDisplayFunc(display); // display func

glutKeyboardFunc(key); // keyboard func: input

    glutMouseFunc(mous); // when mouse first clicked: input

    glutMotionFunc(mouser); // when mouse is moved: input


glutMainLoop(); // loop

    return 0;       // return statement

}
5.3 Initialization

The initialization function sets all of the initial data to be used in the program. The Quadric
object qobj is given default value to be modified and used later. this quadric object is sent
to specific shape display functions

void myinit(void) // initialization
{

qobj = gluNewQuadric();

//gluQuadricDrawStyle(qobj, GLU_FILL);  
gluQuadricOrientation(qobj, GLU_INSIDE);  
gluQuadricNormals(qobj, GLU_FLAT); // GLU_FLAT GLU_SMOOTH GLU_NONE

    // original values
    GLfloat light_ambient[] = {.2, .2, .2, 1.0}; // .3 .3 .3 1.0
    GLfloat light_diffuse[] = {.9, .9, .9, 1.0}; // 1 1 1 1
    GLfloat light_specular[] = {.0, .0, .0, 1.0}; // 1 1 1 1
    GLfloat light_position[] = {0.0, 4.0, 10.0, 0.0};

    //************** Mr. Hyatt’s light initialization statements**************

    GLfloat mat_specular[] = {.5, .5, .5, 1.0}; // .5, .5, .5, 1.0
    GLfloat mat_diffuse[] = {0.8, 0.8, 0.8, 1.0};
GLfloat mat_shininess[] = {30.0}; // 30

glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);

// Material Properties
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);

// Enable Various Components
glEnable(GL_COLOR_MATERIAL);
glEnable(GL_LIGHT0);
glEnable(GL_DEPTH_TEST);
glEnable(GL_LIGHTING);

glColorMaterial(GL_FRONT, GL_DIFFUSE);

//************************************************************************ END************************************************************************
// my data initiation

// joints are largest and longest at the base of the middle finger and
// get smaller as you move away from that digit

glColor3f(1.0, 1.0, 0.0);  // yellow

// glColor3f(0, 0, 1);    // test colors

// glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);

for(int x=0; x<4; x++)
for(int y=0; y<4; y++)
{
  joint[x][y] = .95-.05*abs(x-1)-.1*y;
}

for(int x=0; x<4; x++)
for(int y=0; y<3; y++)
{
  dist[x][y] = 2.2-.4*y-.05*pow((x-1), 2);
  angle[x][y]=15;
}

for(int y=0; y<3; y++) // for the thumb-diff proportions from the rest of the hand
{ 
  dist[4][y] = 2.5-.5*double(y);
  angle[4][y] = 15;
  joint[4][y] = 1.1-.1*y;
  spread[y] = 10*y;
}

joint[4][3] = .7;
angle[4][0] = 40;

The final for loops are used to set the initial values of my arrays. These arrays are used for different

5.4 Helper Functions

The functions are defined early in the code and then used throughout the rest of the program to make the code later simpler and more organized.

double abs(double x) // returns the absolute value of the integer entered
{
  if(x<0)
    return (x*-1);
  return x;
}
int pow(int base, int exp) // returns base ^ exp
{
    if(!exp)
        return 1;
    return base*pow(base, exp-1);
}

void rotx(int num)
{
    glRotatef(num, 1.0, 0.0, 0.0);
}

abs returns the absolute value of the integer sent to it. pow returns, recursively, the value of base raised to the exp power. rotx, in openGL, rotates the current matrix num degrees around the x axis of the picture. this only saves space in the code, not lines, and makes the calls easier to create.
5.5 reshape

This function is used so, when the window is resized in openGL, the picture changes its dimensions as well to fill the same proportions of the window.

```
void myReshape(int w, int h)    //****** Mr. Hyatt's reshape function
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    gluPerspective(40.0, (GLfloat)w / (GLfloat)h, 0.1, 120.0);
    glMatrixMode(GL_MODELVIEW);
}
```

5.6 display

This function is the main function for display of my model. It calls a lot of different functions in order to dcreate the entire hand.

```
void display()
{
```
gluPerspective( 40.0, 1.0 , 0.1, 120.0); // set the perspective 40.0, 1.0, 0.1, 120.0

glMatrixMode(GL_MODELVIEW); // set matrix mode

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // clear the screen

glLoadIdentity(); // load identity matrix

gluLookAt(0.0, 0.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0); // change camera position

glRotatef(double(xrot), 0, 1, 0); // rotate the screen to supply for

glRotatef(double(yrot), 1, 0, 0); // multiple different viewing angles

glPushMatrix();
palm(); // draw the palm of the hand

for(int x=0; x<4; x++)
    finger(x, 2.5-1.66666666666666666666666666666666*x, 2.85-1.0*pow((x-1), 2)); // draw each finger at a different location

thumb(); // draws the thumb

glPopMatrix();

flush(); // data to screen

    glutSwapBuffers(); // buffer the picture

    glutPostRedisplay(); // display
}

13
A lot of the early function calls are just mandatory in order to create the right conditions to start making the object to be displayed. `gluLookAt` sets the camera's position and view. The first three coordinates are the positions of the camera. The second set of three numbers is where the camera is looking. The final set of three numbers tells the computer what directions is up for the camera. This sets the camera position and angle. The two rotation functions is a way for the user to look at different parts of the hand from different angles. Exactly how this works will be covered in the mouse function. A matrix is then pushed to draw on. Then, the palm function is called that draws the palm of the hand. A for loop runs to create the four different fingers at different positions according to a formula. This way allows me to have only one function for the drawing of the finger instead of having a function for each finger. I could have obtained a few hundred extra lines of code by doing this, but I decided organization and compaction was a better idea. The matrix is then popped and flushed to the screen with buffering to make changes of images smooth.

5.7 palm

```cpp
void palm() // draws the palm of the hand
{

glColor3f(1.0, 1.0, 0.0);


glPushMatrix();
```

14
glutSolidTorus(1.0, 2.5, 20, 20); // the circular part of the palm

glPopMatrix();

glPushMatrix();

glTranslatef(0, 0, .7);

 gluDisk(glNewQuadric(), 0, 2.5, F, F);

glPopMatrix();

glPushMatrix();

glTranslatef(0, 0, .49);

//glRotatef(180, 1.0, 0, 0);

//gluQuadricDrawStyle(qobj, GLU_FILL);

//gluQuadricOrientation(qobj, GLU_INSIDE);

//gluQuadricNormals(qobj, GLU_FLAT); // GLU_FLAT GLU_SMOOTH GLU_NONE

//gluQuadricOrientation(qobj, GLU_INSIDE);

//gluCylinder(glNewQuadric(), 0, 2.5, .5, F, F);

   glPopMatrix();

   glPushMatrix();

   glTranslatef(0, 0, -.9);

gluDisk(qobj, 0, 2.5, F, F);

glPopMatrix();

}
This function, obviously, draws the palm of the hand. The torus is the curcular rim of the hand. The two disks that come next complete the flat portions of the hand inside the torus. The first call uses a new quadric because the normals are toward the screen, or out on the near portion. The second call uses qobj because it has reversed normals. because this is the back of the hand, this results again in outward normals. The middle section of code was an attempt to smooth the line between the torus and the 2 planes. This would have been done with a shallow cone that only appears as a filler where the torus and planes meet, making a smoother transition. For a reason that I could not figure out, the normals would not reverse to the inside of the cone, something that was necessary in order for the cone to appear the right color. I tried many different modifir functions that the OpenGL book said would work, but none did. I eventually scrapped this idea and lived with a slightly less realistic model.

5.8 finger

void finger(int test, double x, double y) // draws, you guessed it, a finger {

glPushMatrix();

    rotx(-90);

    glTranslatef(2.5-1.6666666*test, 0, 0);    // move to x position of finger

double tempy = 2.85-.15*pow((test-1), 2);

    if(cyl[test])   // if told to (cyl == true)

        gluCylinder(gluNewQuadric(), 1.0, joint[test][0], tempy, F, F); // draws a cylinder
// the base of the knuckle to the side of the hand

glPushMatrix();

glTranslatef(0, 0, tempy); // move to y position of finger

glutSolidSphere(joint[test][0], F, F); // draws the knuckle

glPushMatrix();

    //glRotatef(-5, 0, 0, 1.0);
    rotx(angle[test][0]);

    gluCylinder(gluNewQuadric(), joint[test][0], joint[test][1], dist[test][0], F, F); //

        glTranslatef(0, 0, dist[test][0]); // | first joint

        glutSolidSphere(joint[test][1], F, F); // -

        glPushMatrix();

    rotx(angle[test][1]); // -

        gluCylinder(gluNewQuadric(), joint[test][1], joint[test][2], dist[test][1], F, F);

            // | second joint

        glTranslatef(0, 0, dist[test][1]); // |

        glutSolidSphere(joint[test][2], F, F); // -

        glPushMatrix();

            rotx(angle[test][2]); // -

            gluCylinder(gluNewQuadric(), joint[test][2], joint[test][3], dist[test][2], F, F);

                // | final digit

        glTranslatef(0, 0, dist[test][2]); // |
```cpp
    glutSolidSphere(joint[test][3], F, F); // -
    glPopMatrix();
    glPopMatrix();
    glPopMatrix();
    glPopMatrix();
    glPopMatrix();
    glPopMatrix();

} 

The finger was the most complicated functions simply because it had to cover for all 4.
The number test is used to differentiate between the different fingers. double x and y are
currently not used, although they would have been the position variables in earlier versions.
Now, tough, the position is calculated in the finger function using the number test, or which
finger is being drawn. This function draws the finger starting at the position calculated. cyl[]
is an array that decides whether or not a cylinder is drawn to the palm from the knuckle.
This is done for the 2 outside fingers in order to make the hand appear realistic. The first
2 translate calls moves the drawing position to the start of the hand. The first sphere is
drawn as the knuckle with a size determined by the array joint[]. every knuckle after it is
determined through sixe by this 2 dimensional array. rotx is called to rotate the finger, with
respect to the x asis, as much as the array angle[] tells it to. This is the bend in the finger
itself. this array is modifiable in order to allow for the user to change the bend in the fingers
and move the hand. the cylinder is then drawn that is the digit itself. The radius of the
bottom of the finger is the same as the knuckle at the bas of the digit, and the raduis of the

top of the digit is the same as the radius of the joint at the top of the digit. This amakes for smooth continuous bends in the finger without overy large knuckles. The position of the matrix, or where things are drawn, is then moved to the end of the digit, or cylinder. This process is repeated for all of the 3 digits of each finger, which by definition of the initialization of the arrays, get smaller the firther from the hand you get. A final sphere is drawn at the end of the third digit to cap the finger, so it doesnt appear hollow. The middle finger is the larges, both in length and in radius. The ring and index finger are the same size, and the pinkie is the smallest. This is determined through the earlier inital values given in the initialization function. At the end of the function, all of the matrices used for drawing are popped, kind of the way of ending that particular reference which built upon each other as the drawing moved out along the finger. These matrices create a reference that moves so that everything is drawn at its origin, which is moved to the different parts of the screen and rotated as to create the image desired. popping them resets the drawing perspective to the original coordinates of (0, 0, 0). This allows for other objects to be drawn later.

5.9 thumb

void thumb() // creates the thumb
{
    int bob = -40;
    double radian = double(bob) / 180.0 * pi;
    glPushMatrix();
glTranslatef(cos(radian) * 2.5, sin(radian) * 2.5, 0);

glutSolidSphere(joint[4][0], F, F);

glPushMatrix();

glRotatef(90, 0, 1, 0);

glRotatef(angle[4][0]-90, 1, 0, 0);  // move away from hand

glRotatef(-spread[3], 0, 1, 0);  // move out of palm plane

gluCylinder(gluNewQuadric(), joint[4][0], joint[4][1], dist[4][0], F, F);

glTranslatef(0, 0, dist[4][0]);

gluSphere(gluNewQuadric(), joint[4][1], F, F);

glPushMatrix();

glRotatef(-1*angle[4][1], 1, 0, 0);

gluCylinder(gluNewQuadric(), joint[4][1], joint[4][2], dist[4][1], F, F);

glTranslatef(0, 0, dist[4][1]);

gluSphere(gluNewQuadric(), joint[4][2], F, F);

glPushMatrix();

glRotatef(-1*angle[4][2], 1, 0, 0);

gluCylinder(gluNewQuadric(), joint[4][2], joint[4][3], dist[4][2], F, F);

glTranslatef(0, 0, dist[4][2]);

gluSphere(gluNewQuadric(), joint[4][3], F, F);

glPopMatrix();
glPopMatrix();

      glPopMatrix();
}

The thumb was a very confusing thing to make. The thumb is just somewhat more mobile and modifiable than the fingers that it was a real challenge. bob, Which will be replaced later with spread as it becomes a capability, determines the angle of rotation away from the hand of the thumb. Spread will be used in the fingers to determine how far apart they are spread from each other. Other than that, the thumb is drawn very similarly to the fingers, just a bit broader and not in a straight line up from the hand. Instead, it is drawn away from the hand, and looks very peculiar without the flap of skin connecting it to the palm like in a real human hand. Everything else can be correlated to the finger function.

5.10 mouse

void mouse(int button, int state, int x, int y) // when mouse first clicked: set initial // of holder variables
{
    xo=x;
    yo=y;
}

void mouser( int x, int y) // when mouse moved, change actual rotation variables
// and reset the holder variables
{
  xrot+=x-x0;
  yrot+=yo-y;
  x0=x;
  y0=y;
}

These functions took me a while to get exactly how I wanted them, and the took a few more global variables than I expected. The initial values of the mouse position are set when the mouse is first clicked to x0 and y0. Then, as the mouse moves, the rot variables, which is what determines the rotation of the display, are reset with every tiny change. Of course, since the picture is redrawn with every small change, the x0 and y0 values are also reset. This allows for constant and smooth changes in the picture rotation. Generally, by clicking and dragging the mouse, the image will rotate around so it can be viewed from all angles.

5.11 keyboard

bool isnum(unsigned char key)
{
  return (int(key) < 59 & int(key) > 47);

  /*(key == '0' || // if its a number
key == '1' ||
key == '2' ||
key == '3' ||
key == '4' ||
key == '5' ||
key == '6' ||
key == '7' ||
key == '8' ||
key == '9')*/
}

int tonum(unsigned char key)
{
return ((int(key) - 48) * 10);
/*switch(key){
case '0': return 0; break;
case '1': return 10; break;
case '2': return 20; break;
case '3': return 30; break;
case '4': return 40; break;
case '5': return 50; break;
}
case '6': return 60; break;
case '7': return 70; break;
case '8': return 80; break;
case '9': return 90; break;
}*/
}

void key(unsigned char key, int g, int h) // my keyboard function
{
    int temp = int(key);
    if(isnum(key)) // if key is a number
    {
        temp = tonum(key); // set temp
        for(int x=0; x<5; x++)
            for(int y=0; y<3; y++)
                angle[x][y]=temp; // set angle of all joints
    }
    else if(((key>65)&&(key<=90))||((key>=97)&&(key<=122))) // if a letter is pressed
    {
        motion(key);
    }
This is my main keyboard function. It is also very complicated because the keyboard is used to modify the hand position for every little thing. If the key pressed is a number (the isnum function returns true if it is and false if it isn't) then temp equals the number of the key times ten (tonum returns the number of the key). Using askii, I was able to come up with a formula that does exactly the same thing as the commented switch statement, returning the current key. When a number is pressed, the angle of every single joint is set to the number temp, or ten times the number pressed. This makes for easy modification of the entire hand quickly. Otherwise, if the key pressed is a number, it is sent to the motion function.

5.12 motion

void add(int& bob)
{
    if(bob < 90)
        bob++;
}

void sub(int& bob)
if (bob > 0)
    bob--;
case 'e': add(angle[1][2]); break;  // middle

case 'E': sub(angle[1][2]); break;

case 'd': add(angle[1][1]); break;

case 'D': sub(angle[1][1]); break;

case 'c': add(angle[1][0]); break;

case 'C': sub(angle[1][0]); break;

case 'w': add(angle[0][2]); break;  // index

case 'W': sub(angle[0][2]); break;

case 's': add(angle[0][1]); break;

case 'S': sub(angle[0][1]); break;

case 'x': add(angle[0][0]); break;

case 'X': sub(angle[0][0]); break;

case 'q': add(angle[4][2]); break;  // thumb

case 'Q': sub(angle[4][2]); break;

case 'a': add(angle[4][1]); break;

case 'A': sub(angle[4][1]); break;

case 'z': add(angle[4][0]); break;

case 'Z': sub(angle[4][0]); break;

case 'y': add(spread[4]); break;  // thumb spread

case 'Y': sub(spread[4]); break;

case 'u': add(spread[3]); break;  // index spread
case 'U': sub(spread[3]); break;
case 'i': add(spread[2]); break; // middle spread

```c
    case 'I': sub(spread[2]); break;
```  

```c
    case 'o': add(spread[1]); break; // ring spread
    case 'O': sub(spread[1]); break;
    case 'p': add(spread[0]); break; // pinkie spread
    case 'P': sub(spread[0]); break;

    }
```  

```c
} }
```  

Motion makes small changes (one degree at a time) to the finger angles. Each key is a shortcut to a certain joint. Imagine placing your right hand on the keyboard, aligned to the left so your thumb fills the far left column of keys. Those three keys, q, a, z, change the angle of the thumb. The next column change the index finger, etc, until t, g, b, change the pinkie angles. When the key is pressed, it reduces the angle of the joint, making a fist. When the capital of the same letter (shift + that letter) is pressed, the joint strightens, both one degree at a time. That is what the helper functions add and sub do. They are called, and add or subtract one degree to the sent joint. They also limit the angles of the joints to between straight (no overextension) and 90 degrees bent. The spread is modified in a similar fasion through the keys y, u, i, o, and p.

28
5.13 conclusion

The total code results in a functional model of a hand. The model is realistic in appearance. It is modifiable in large increments for easy manipulation. It can be contorted to very specific positions, capable of every contortion that a real human hand can accomplish. My goal was accomplished through simplification of code and compaction for organizational purposes.

6 Acknowledgements

I would like to acknowledge Joey Turner for some suggestions he had made for my program as far as the bounds and capabilities of OpenGL. He tried to help me implement the transition between the torus and the plane in the palm, but became as frustrated as I did. I would also like to recognize George Ding and David Sterling for their ridicule of my poor models and compliments to the final realistic model.

7 References

References: 1. http://www.tjhsst.edu/ draber/supercomp/Helpinghand.html The tech lab of a previous pair of students. I was told of this project after I had chosen my tech lab project. Mr Latimer told me about it, so I checked it out. If nothing else, it gave me an idea of what not to make, in order to keep from copying and also because I thought I could make mine appear more realistic. 2. http://www.cs.ubc.ca/nest/magic/projects/hands/home This site let me know of all of the other people with similar project ideas. This site gave me some very
good background information as well as links to many other valuable resources in determining
good background and purpose for my project. 3. http://fas.sfu.ca/cs/people/ResearchStaff/amulder/personal/vr

/www.merl.com/papers/TR94–24/