Appendix E - UNIVERSITY HONORS PROGRAM
SENIOR PROJECT - APPROVAL

Name: Danielle L. Smith

College: A & S Department: Computer Science

Faculty Mentor: Dr. Jian Huang

PROJECT TITLE: Animated Graphics Scene: Ghostly Happenings

I have reviewed this completed senior honors thesis with this student and certify that it is a project commensurate with honors level undergraduate research in this field.

Signed: _____, Faculty Mentor

Date: Dec. 11, 2002

General Assessment - please provide a short paragraph that highlights the most significant features of the project.

Comments (Optional):

I basically programmed and assembled a graphics scene with bump mapping, texture mapping and animation. There are objects that move around in the scene. The actual program also consists of function to load .off, .obj, and .ply files, a display, and severeral other functions to perform the techniques listed above.

Animated Graphics Scene:

Ghostly Happenings

By

Danielle L. Smith

Throughout the entire course, I used a structure called "OBJ" to represent each 3d object rendered by my code. The object started out with only a few items in it to hold the vertices, edges, faces, and the information related to the number of each. This structure eventually evolved into a much larger, more developed one. Its components are as follows:

Name	Type	Description
vertices	int	Number of vertices
faces	int	Number of faces
v_list	double**	Dynamically allocated list of vertices
nrmls	double **	Dynamically allocated list of normals
f_list	int**	Dynamically allocated list of faces
text	double**	Dynamically allocated list of texture coordinates
file	char [25]	File name of object
div	int	Number to scale object size by
tfile	char[40]	File name of object's texture

Table 1- object structure

This data structure was mostly initialized and created in the danparse.c file. This file's sole purpose is to read in the object's file and to take in the individual information of the object that would allow the renderer to render the object. It also contained programs to do some initial editing of the object in order to make it easier on my display function. Danparse's main function is load_file.

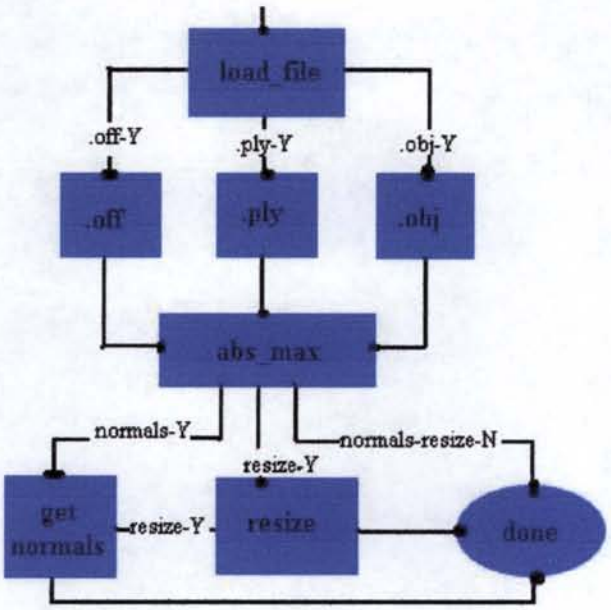

Table 2- Flow chart of load_file

This function in its original form only took a file name. When I began to implement multiple objects, it took the index of each object and its filename, which, by then, had become a portion of the structure. One might wonder why I allowed the filename to remain as input to the function when access was already available through the index of the object. I'll just say that it was easier to just add the open and closed brackets to the end of each object name. At the time, I decided that keeping the changes to a minimum would be the best approach. When anyone goes in and changes too many things, the balance of the program can be dramatically disrupted and this can cause more trouble than it's worth. The **load_file** program consisted of functions that examined the file name and the first few lines of the code and sent it to the separate functions that read each format. I realized later that this caused a bug. Only the **ply_parse** and **off_parse** functions' first lines consisted of an indicator that allowed the reader of the file to know what type each file was; and reading in the first few lines until I reached some writing

caused a problem with reading in my .obj files. The .obj files did not have to have a line that indicated what the file type was, thus when trying to divine the file type, if the file was a .obj file, I ended up discarding the first few lines.

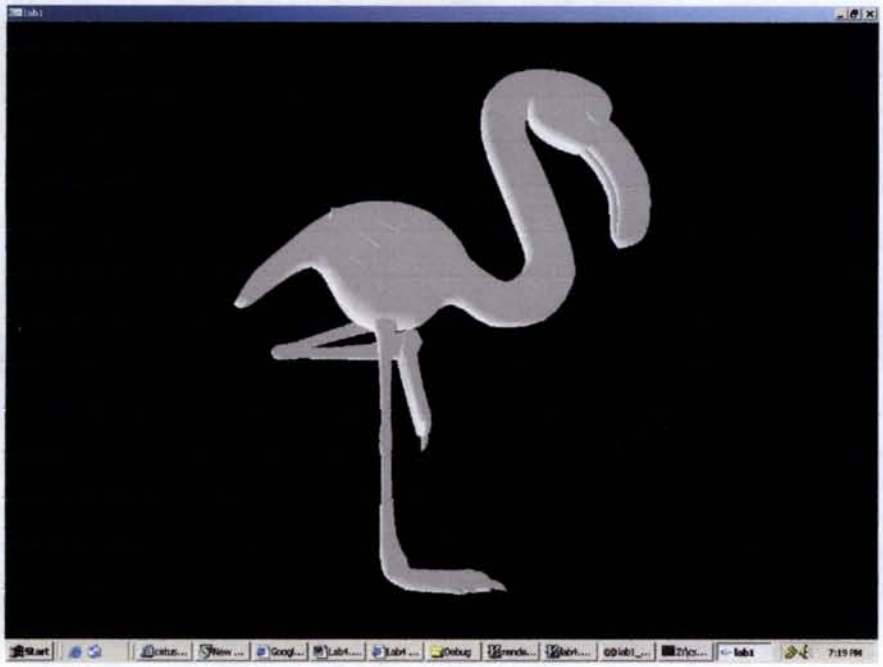

Figure 1- Flamingo.obj

These lines were sometimes vertex information and sometimes comments by the writer, so sometimes the files worked and sometimes they did not. I could have changed my code to read the file name and look for the extension, but instead I just added a comment to the first line of all my .obj files. The **fndabsmax** function found the absolute value of the largest vertex, abs_max. Another function, **crssnrml**, found the normals. Many of the .obj files had their own normals, so if the **obj_parse** properly read in all of the normals then the separate function to find the normals was skipped. I also found that

many of the objects were too large and extended far beyond the immediate range of the "camera" and far beyond the "camera" itself. This was caused by vertices that were very large and these large numbers resulted in distorted objects. At first, I assumed that the problem was, as it often is, with my parse functions. Then when I realized what the problem was, I wrote a function that used the object[*].div to fix the problem. I set div to be some number. The abs_max is divided by div and then, each vertex is then divided by this number. These calculations are done in the **correct** function and are bypassed if div is set to 0. This is the final function that could be called while loading geometry files and after this point the control is returned to the main function.

In labs one through three, I spend much of the time changing, inserting, and deleting lines, in order to see their effect on the rendering of function. Loading in the geometry was a simple matter of brute-force programming in C. The specifics are described in the previous section. In setting the transformation matrices and manipulating the graphical user interface to render different views of my scenes is where I had first big problem. The code ran and the objects turned, but I noticed that it changed the object's local coordinate system as well. For example, this caused the inherent understanding that the right arrow key turned the object counter clockwise to be disrupted. The right arrow key could easily turn the object back in the negative x direction one time and in the counter clockwise direction the next. This was fixed by changing the order of my glRotate and glTranslate calls. Changing the fulstrum angle was the method I used to zoom towards and away from the object. I also added a feature to the "Home" button, to restore the original Modelview matrix and fulstrum. This construct was added as a result of the earlier mentioned rotation and translation problems.

The lighting conditions and changes gave me the least problems in the process of coding for this class. I just set and enabled the lights and smooth shading in the initializing function. In order to change the lighting to reflect different types of materials, I loaded a list of the ambient, diffuse, and specular components of each material from a file into a 2 -dimensional array. Then, I switched from material to material by pressing "m". At the end of the list then the list array index was set to 0 and the original conditions were restored.



Figure-2 Materials (r-obsidian, l- black plastic)

Basic polygon texture was fairly simple. I ended up making it harder than it should have been, but eventually got the hang of it. The hardest portion was determining what to send to the read_texture function and what typed to declare each variable as. During the 3d texture portion of lab2, I spend many hours trying to fix a problem caused by incorrect declarations. I also have not figured out if there is a way to simply color and object and texture another at the same time. When I tried this, it often resulted in a colored texture.

Environment mapping, cube and sphere, were only a matter of changing a few lines of code and sending in different image files. I had some problems before I discovered the magic of glTexGeni. In procedural mapping where we had to generate our own texture coordinates the problems came in the math. It was hard to visualize the way matrices were to be multiplied when looking at code that deals with individual numbers and not with the whole picture. The formula provided by the book was incomplete as was my memories of matrix algebra. I programmed the formulas that were in the book in order to get texture coordinates and I ran my code, after a small debugging period. I realized that I had woefully fallen short of the beautiful texture coordinates generated by openGL. The answer did not eventually dawn on me, I received a bit of help from one of my colleagues. After the proper inversions of different matrices before multiplying them with other matrices the function worked.



Figure 3 - Environment mapping

Embossed bump mapping was a bit harder than I thought it would be. My first problems arose with trying to get the glaux.h library. When I added it in just like the gl.h library was included, the complier still could not find it. After changing it several times and then putting it back to the way it was originally, it worked. I am still not sure why it would not work to begin with, but I was glad when it finally did compile. Then I focused on the textures themselves. The textures are read in and then nearest filtered, linear filtered and MIP mapped. This is done to the base, bump, and inverted bump textures. The inverted bump texture was created by subtracting the different color components of each texel for the image from 255. The display function only calls the bump mapping function and glSwapBuffers.

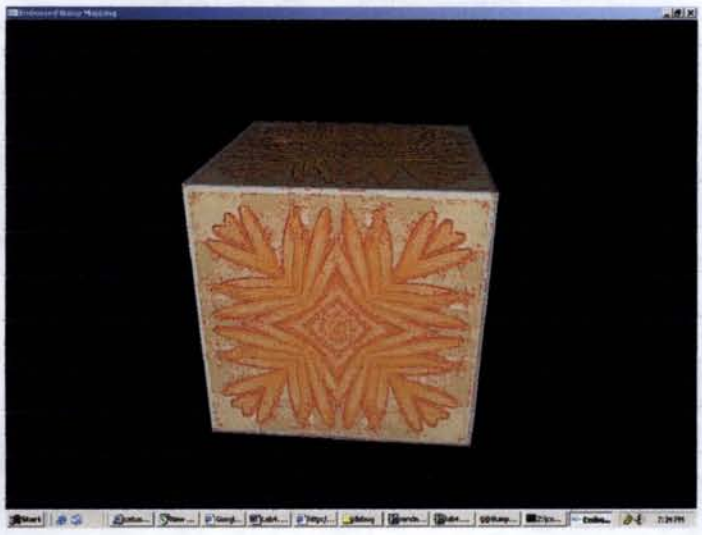

Figure 4- Embossed Bump Mapping

Anti-aliasing with the A-Buffer was easy to do, but the hard parts were finding out what I had to do. It took me a long time to find the correct jittering codes. It also took

me a while to realize that I had to add the A-Buffer. I think that it was distant in my mind, but I had not yet consciously realized it. When I did get everything running to a point where I could actually see results on the screen, I realized that something was not working correctly. I later found out that it was that I had to change the textures. So, had I know exactly what I needed to do, this would have taken me ten to fifteen minutes. I also had a bit of a start when I realized that it took some time after you switch between MIP mapping and A-buffer for MIP mapping to take effect.

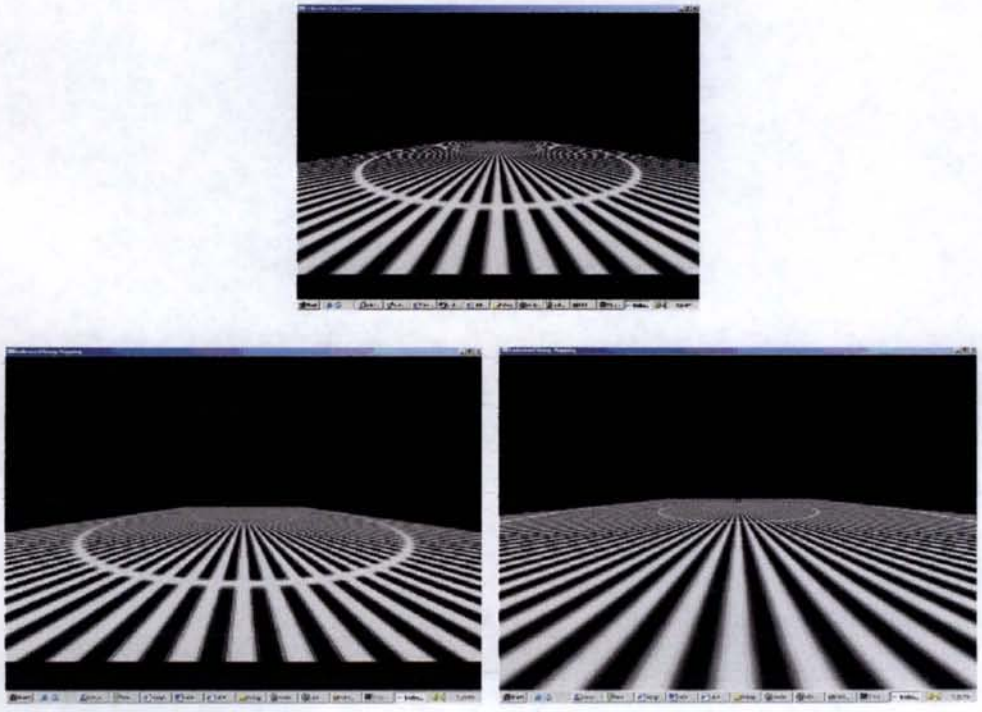

Figure 5- Anti-Aliasing (top- original, bottom- r, MIP mapping ;l, A-buffer)

I decided to render an interesting scene with bump mapped walls for lab four. The room consists of eight objects with different textures. I used display lists for six of the objects. I spent a considerable amount of time looking for the few objects that I have

in the room. I made the objects that I could not find or could not get to render correctly. All of the made objects are a series of cubes. It took a good amount of time to get the coordinates for the correct positioning of these made objects. I spent even longer looking for textures. The problem was my fault because I did not realize that the size of each texture image had to be a power of two. Because of this, I spent many hours looking for the textures that I liked and then discarded them because they did not work. Looking back at this, I can laugh at myself, but while I was working on this lab, it was very painful. The big programming challenge was in learning how to bump map the walls of my room while making the floors texture mapped. Once I get something working well, I tend to lean towards not changing it much if at all. So, my problem became how to fix things without too much editing. I first tried to just add a texture mapped floor over the bump mapped floor. I realize that this is an inefficient method, but I wanted to make sure that my code would still work after the changes. This method, did not work. It seems that my floors did not appear at all, neither bump nor texture mapped. In retrospect, I believe that the floor and ceiling were there, but not visible due to the black background. Then I tried to render the floor in color both before and after the room was rendered with bump mapping. This resulted in a colored bump map with still no floor or ceiling. Then I noticed that when I turned off the bump mapping, the floor and ceiling appeared. This made me realize that I must change my bump mapping and cube creating functions. So, I deleted the floor and ceiling from the bump mapping function to make the cube without the floor and ceiling, thinking that if the floor and ceiling were not bump mapped then I could texture map them myself. So, I added the **glBindTextures** function to the function that created the cube. I also separated the floor and ceiling into their own **glBegin** and

glEnd. I also enabled the lighting, in order for the floor and ceilings to be rendered with light. This worked well. The floors were texture mapped every time the cube function was called.

Then came the time to render the objects in my room. As stated in the above section , I changed the load_file function and related functions to work with an array of my object structures. In the main function I set the filenames and textures for each object, load the files, and set the object[*].div number for each object. Then in the init function, I ran a function to load all of the textures that my objects used. I put them into a separate function because it is easier to change and fix one function without editing the others. Also, in the init, I ran a function to build my display lists. I decided to create two more functions to position my objects around the room. The first function did translating, scaling, and rotating for each object. The second did the opposite actions for each object in order to negate them when the time came to redo the room in the display function. Each different series of translations, rotations, and scales for each individual object is keyed on the index number of my array of structures that holds the object.

```
Begin build display list function
     Generate list (8)
     Begin for from 0 to 6
     New lists
          Placing function
          Glbegin
               Load vertex coordinates
               Load texture coordinates
          GlEnd
          Reverse Placing function
     End lists
End build display list function
```

Example 1 – Pseudo code for building display lists

In order to get every thing animating correctly, I ended up translating everything that I wanted to move before they were actually rendered and then rotating them about their original y-axis in object space. When the "a" key is pushed then it activates a flag that runs the function that I use to animate the objects. During a given period of time, while the other objects are still moving, two more objects pop onto the screen. These objects are flagged to appear at a certain time in the rotation of the moving objects. It appears that one is morphing into the other.

```
Begin function to draw object
     Enable generation of texture coordinates
     Draw stationary objects
     If s and w flags TRUE
          Draw woman
          And skeleton
     If animation flag TRUE
          Rotate
          Translate
     Render non stationary objects
     If animation flag TRUE
          Reverse Translate
          Reverse Rotate
     Push original Modelview matrix on stack
End draw function
```

Example 2- Psuedo Code for animating objects

The code above is run in the function that bump maps the room, and that function is run in the display function.

The timing for my code is pretty interesting. When it first begins in the smaller screen the timing is in the high eighty's. Then when I begin the animation portion of the code it drops to mid forty's around 46 and 58. Finally, if it's animating and in the full screen it drops to the high teens and low twenty's. I can only guess how bad it would be had I not used display lists. When the animation begins, the program has to draw two

more objects. These two are the ones that pop in and out of the scene. Also, in full screen, there are more pixels to render, so I can understand the large jump in performance levels.

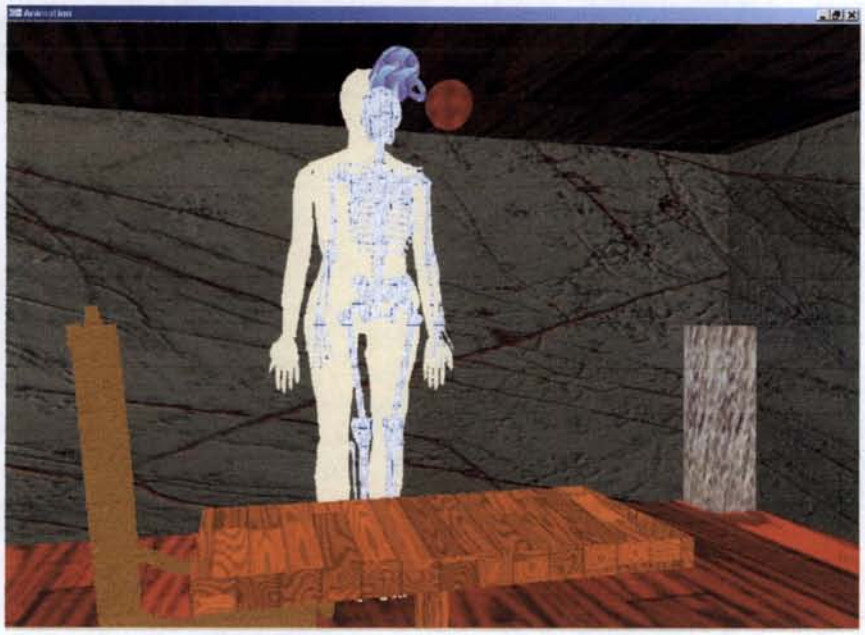

Figure 6- Lab 4 (Animation)

Discussion

As to changing my code to work with reflections, I think that it would only be a matter of a adding the relevant lines to my init, adding the stencil buffer in my main program, and putting all the reflection portions in a function that I can add to my display. My code is a decent practice in modularity. I do not have many long functions and each is a single portion that relates to only one action. Please note that I am not saying that making a mirror would be simple, merely that adding the proper functions would. In

order to have a scene graph renderer, I would have to change the way I open and display the files. They wouldn't have to be opened or displayed until they are first used. Also, I would have to have some code to put them in a usage graph. I believe that this would take a lot of extra work. Adding another function, as stated earlier would not be much of a problem. I am not sure how I would change my program to work with distorting objects, it currently has no problem with moving objects. I actually did a lot of hacking for this entire course. I believe that my code is not very robust, and I hold up my frames per second results as an illustration.

```
#include "lab4.h"

int maxTexelUnits = 1;
//double zoom = 5.0;
//float X = 0.0, Y = 0.0, xspeed = 0.0, yspeed = 0.0;
//float yrot, xrot;
int  filter = 1;
int  texture[3];
int  bump[3];
int  invbump[3];
int aflag = 0, sflag = 0, wflag = 0;
int emboss = 0;
int fps = 0;
time_t stime, endtime;
GLfloat Orig[16];
GLuint dlists, list[4];

char *basefile = "Z:\\Data\\pinmarbl.rgb";
char *bumpfile = "Z:\\Data\\pinbump.rgb";
GLfloat lmodel_ambient[]    = { 0.2f, 0.2f, 0.2f};              // Ambient Light Is 20% White
GLfloat white_light[]   = { 1.0, 1.0, 1.0};                 // Diffuse Light Is White
GLfloat light_position[]    = { 0.0, 13.0, 0.0};                    // Position Is Somewhat In Fro
nt Of Screen
GLfloat Gray[]      = { 0.5f, 0.5f, 0.5f, 1.0};

char file[25] = "Z:\\plank.rgb";
char tped[25] = "Z:\\Data\\lrock033.rgb";
char text2[25] = "Z:\\plank.rgb";
GLuint tabl, tobj[8], tandb, ped;
GLfloat xrot;                                                   // X Rotation
GLfloat yrot;                                                   // Y Rotation
GLfloat xspeed;                                                 // X Rotation Speed
GLfloat yspeed;                                                 // Y Rotation Speed
GLfloat z=-5.0;
    GLfloat l[4];                                       // Holds Our Lightposition To Be Transform
ed Into Object Space
    GLfloat Minv[16];

PFNGLMULTITEXCOORD1FARBPROC glMultiTexCoord1fARB = NULL;
PFNGLMULTITEXCOORD2FARBPROC glMultiTexCoord2fARB = NULL;
PFNGLMULTITEXCOORD3FARBPROC glMultiTexCoord3fARB = NULL;
PFNGLMULTITEXCOORD4FARBPROC glMultiTexCoord4fARB = NULL;
PFNGLACTIVETEXTUREARBPROC   glActiveTextureARB = NULL;
PFNGLCLIENTACTIVETEXTUREARBPROC glClientActiveTextureARB = NULL;

GLfloat data[]= {

        // Back Face

        1.0f, 0.0f,     -35.0f, -15.0f, -30.0f,
        1.0f, 1.0f,     -35.0f,  15.0f, -30.0f,
        0.0f, 1.0f,      35.0f,  15.0f, -30.0f,
        0.0f, 0.0f,      35.0f, -15.0f, -30.0f,

        // Right Face
        1.0f, 0.0f,      35.0f, -15.0f, -30.0f,
        1.0f, 1.0f,      35.0f,  15.0f, -30.0f,
        0.0f, 1.0f,      35.0f,  15.0f,  30.0f,
        0.0f, 0.0f,      35.0f, -15.0f,  30.0f,

        // Left Face

        0.0f, 0.0f,     -35.0f, -15.0f, -30.0f,
        1.0f, 0.0f,     -35.0f, -15.0f,  30.0f,
        1.0f, 1.0f,     -35.0f,  15.0f,  30.0f,
        0.0f, 1.0f,     -35.0f,  15.0f, -30.0f,

        // Top face
        0.0f, 1.0f,     -35.0f,  15.0f, -30.0f,
        0.0f, 0.0f,     -35.0f,  15.0f,  30.0f,
        1.0f, 0.0f,      35.0f,  15.0f,  30.0f,
        1.0f, 1.0f,      35.0f,  15.0f, -30.0f,
```

```
        // Bottom Face
        1.0f, 1.0f,     -35.0f, -15.0f, -30.0f,
        0.0f, 1.0f,      35.0f, -15.0f, -30.0f,
        0.0f, 0.0f,      35.0f, -15.0f,  30.0f,
        1.0f, 0.0f,     -35.0f, -15.0f,  30.0f,
};



void init(void)
{
    unsigned* image2;
    int width, height, components;

    glClearColor(0.0, 0.0, 0.0, 0.5);
    glShadeModel(GL_SMOOTH);


    glClearDepth(1.0);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);


    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, white_light);
    glLightfv(GL_LIGHT0, GL_AMBIENT, lmodel_ambient);

    glEnable(GL_LIGHT0);

    glEnable(GL_TEXTURE_2D);
    glGetIntegerv(GL_MAX_TEXTURE_UNITS_ARB,&maxTexelUnits);
    glMultiTexCoord1fARB = (PFNGLMULTITEXCOORD1FARBPROC) wglGetProcAddress("glMultiTexCoord1fARB")
;
    glMultiTexCoord2fARB = (PFNGLMULTITEXCOORD2FARBPROC) wglGetProcAddress("glMultiTexCoord2fARB")
;
    glMultiTexCoord3fARB = (PFNGLMULTITEXCOORD3FARBPROC) wglGetProcAddress("glMultiTexCoord3fARB")
;
    glMultiTexCoord4fARB = (PFNGLMULTITEXCOORD4FARBPROC) wglGetProcAddress("glMultiTexCoord4fARB")
;
    glActiveTextureARB   = (PFNGLACTIVETEXTUREARBPROC) wglGetProcAddress("glActiveTextureARB");
    glClientActiveTextureARB= (PFNGLCLIENTACTIVETEXTUREARBPROC) wglGetProcAddress("glClientActiveT
extureARB");
    printf("here 1");
    if(!(LoadGLTextures()))
    {
        printf("PROBLEM LOADING TEXTURES\n");
        exit(1);
    }
    printf("here2 ");


    image2 = read_texture(file, &width, &height, &components);
    glGenTextures(1, &tandb);
    glBindTexture(GL_TEXTURE_2D, tandb);
    glTexImage2D(GL_TEXTURE_2D, 0, components, width,
                 height, 0, GL_RGBA, GL_UNSIGNED_BYTE,
                 image2);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    free(image2);
    objtextures();
    BuildList();
}

void objtextures (void)
{
    unsigned *image;
    int width, height, components, i;
```

```
    printf("here now");
    image = read_texture(text2, &width, &height, &components);
    glGenTextures(1, &tabl);
    glBindTexture(GL_TEXTURE_2D, tabl);
    glTexImage2D(GL_TEXTURE_2D, 0, components, width,
                 height, 0, GL_RGBA, GL_UNSIGNED_BYTE,
                 image);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

    free(image);

    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_LINEAR);
    glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_LINEAR);

    printf("here now");
    image = read_texture(tped, &width, &height, &components);
    glGenTextures(1, &ped);
    glBindTexture(GL_TEXTURE_2D, ped);
    glTexImage2D(GL_TEXTURE_2D, 0, components, width,
                 height, 0, GL_RGBA, GL_UNSIGNED_BYTE,
                 image);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    free(image);

    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_LINEAR);
    glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_LINEAR);

    glGenTextures(8, &tobj[0]);
    for(i = 0; i < numobjs; i++)
    {
        image = read_texture(object[i].tfile, &width, &height, &components);

        glBindTexture(GL_TEXTURE_2D, tobj[i]);
        glTexImage2D(GL_TEXTURE_2D, 0, components, width,
                     height, 0, GL_RGBA, GL_UNSIGNED_BYTE,
                     image);
        if(i == 1)
        {
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);

        }
        else
        {
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

        }

        free(image);

        glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_LINEAR);
        glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_LINEAR);
    }

}


void display(void)
{
    time(&stime);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
    doMesh1TexelUnits();

    if(aflag)
    {
        animate();
    }
    glPopMatrix();

    glFlush();
    glutSwapBuffers();
    time(&endtime);
    if(difftime(stime, endtime) != 0)
    {
        printf("fps: %d\n", fps);
        fps = 0;
    }
    fps++;
}


void reshape(int w, int h)
{
    WIN_SIZE_W = w;
    WIN_SIZE_H = h;
    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (GLfloat) WIN_SIZE_W/(GLfloat) WIN_SIZE_H, 0.1, 100.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

}


void SpecialKeys(int key, int x, int y)
{


    if(key== GLUT_KEY_INSERT)
        exit(1);
    if (key == GLUT_KEY_PAGE_UP)
    {
        z+=0.5f;
    }
    if (key == GLUT_KEY_PAGE_DOWN)
    {
        z-=0.5f;
    }
    if (key == GLUT_KEY_UP)
    {
        xrot = xrot - 5.0;
    }
    if (key == GLUT_KEY_DOWN)
    {
        xrot = xrot + 5.0;
    }
    if (key == GLUT_KEY_RIGHT)
    {
        yrot= yrot + 5.0;
    }
    if (key == GLUT_KEY_LEFT)
    {
        yrot = yrot - 5.0;
    }
}

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

    switch(key)
    {
        case 'e':
```

```
            if(emboss == 0)
                emboss = 1;
            else
                emboss = 0;
            break;
        case 'f':
            filter++;
            if (filter == 3)
                filter = 0;
            printf(" f= %d", filter);
            break;
        case 'a':
                aflag = 1;
            break;
        default:
            break;
    }
}


// Using auxDIBimageLoad's Own Error-Handler!
int LoadGLTextures(){                                          // Load Bitmaps And Convert To
 Textures
    int status, i;                                          // Status Indicator
    static int width, height, comps;
    unsigned *image;                                    // Create Storage Space For The Texture

    image = NULL;
    status = 1;

    // Load The Tile-Bitmap For Base-Texture
    if (image = read_texture(basefile, &width, &height, &comps)) {
        glGenTextures(3, texture);                              // Create Three Textures
        // Create Nearest Filtered Texture
        glBindTexture(GL_TEXTURE_2D, texture[0]);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
        glTexImage2D(GL_TEXTURE_2D, 0, comps, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);
        //                          ========
        // Use GL_RGB8 Instead Of "3" In glTeximage2D. Also Defined By GL: GL_RGBA8 Etc.
        // NEW: Now Creating GL_RGBA8 Textures, Alpha Is 1.0 Where Not Specified By Format.

        // Create Linear Filtered Texture
        glBindTexture(GL_TEXTURE_2D, texture[1]);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
        glTexImage2D(GL_TEXTURE_2D, 0, comps, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);

        // Create MipMapped Texture
        glBindTexture(GL_TEXTURE_2D,texture[2]);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR_MIPMAP_NEAREST);
        gluBuild2DMipmaps(GL_TEXTURE_2D, comps, width, height, GL_RGBA, GL_UNSIGNED_BYTE, image);
    }
    else status=0;
    if (image) {                                                // If Texture Exists
        free (image);                       // If Texture image Exists
        image = NULL;
    }

    // Load The Bumpmaps
    if (image = read_texture(bumpfile, &width, &height, &comps)) {
        printf("here 3");
        glPixelTransferf(GL_RED_SCALE,0.5);                     // Scale RGB By 50%, So That We Ha
ve Only
        glPixelTransferf(GL_GREEN_SCALE,0.5);                       // Half Intenstity
        glPixelTransferf(GL_BLUE_SCALE,0.5);

        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP);  // No Wrapping, Please!
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP);
        glTexParameterfv(GL_TEXTURE_2D,GL_TEXTURE_BORDER_COLOR,Gray);

        glGenTextures(3, bump);                                 // Create Three Textures
```

```
        // Create Nearest Filtered Texture
        glBindTexture(GL_TEXTURE_2D, bump[0]);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
        glTexImage2D(GL_TEXTURE_2D, 0, comps, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);

        // Create Linear Filtered Texture
        glBindTexture(GL_TEXTURE_2D, bump[1]);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
        glTexImage2D(GL_TEXTURE_2D, 0, comps, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);

        // Create MipMapped Texture
        glBindTexture(GL_TEXTURE_2D, bump[2]);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR_MIPMAP_NEAREST);
        gluBuild2DMipmaps(GL_TEXTURE_2D, comps, width, height, GL_RGBA, GL_UNSIGNED_BYTE, image);

        for ( i=0; i<(width*height); i++)// Invert The Bumpmap
            image[i]=255-image[i];

        glGenTextures(3, invbump);                              // Create Three Textures

        // Create Nearest Filtered Texture
        glBindTexture(GL_TEXTURE_2D, invbump[0]);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
        glTexImage2D(GL_TEXTURE_2D, 0, comps, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);

        // Create Linear Filtered Texture
        glBindTexture(GL_TEXTURE_2D, invbump[1]);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
        glTexImage2D(GL_TEXTURE_2D, 0, comps, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);

        // Create MipMapped Texture
        glBindTexture(GL_TEXTURE_2D, invbump[2]);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR_MIPMAP_NEAREST);
        gluBuild2DMipmaps(GL_TEXTURE_2D, comps, width, height, GL_RGBA, GL_UNSIGNED_BYTE, image);

        glPixelTransferf(GL_RED_SCALE,1.0);             // Scale RGB Back To 100% Again
        glPixelTransferf(GL_GREEN_SCALE,1.0);
        glPixelTransferf(GL_BLUE_SCALE,1.0);
    }
    else status=0;
    if (image) {                                            // If Texture Exists
        free(image);
    }

        glPopMatrix();
    glFlush();

    return(status);                                         // Return The Status
}


void doCube (void) {
    int i;

    glBegin(GL_QUADS);
        // Back Face
        glNormal3f( 0.0, 0.0, 1.0);
        for (i=0; i<4; i++) {
            glTexCoord2f(data[5*i],data[5*i+1]);
            glVertex3f(data[5*i+2],data[5*i+3],data[5*i+4]);
        }
        // Right Face
        glNormal3f( -1.0, 0.0, 0.0);
        for (i=4; i<8; i++) {
            glTexCoord2f(data[5*i],data[5*i+1]);
            glVertex3f(data[5*i+2],data[5*i+3],data[5*i+4]);
```

```
        }
        // Left Face
        glNormal3f( 1.0, 0.0, 0.0);
        for (i=8; i<12; i++) {
            glTexCoord2f(data[5*i],data[5*i+1]);
            glVertex3f(data[5*i+2],data[5*i+3],data[5*i+4]);
        }
        glEnd();

        glBindTexture(GL_TEXTURE_2D, tandb);
        glBegin(GL_QUADS);
        glNormal3f( 0.0, -1.0, 0.0);
        for (i=12; i<16; i++) {
            glTexCoord2f(data[5*i],data[5*i+1]);
            glVertex3f(data[5*i+2],data[5*i+3],data[5*i+4]);
        }

        glNormal3f( 0.0, 1.0, 0.0);
        for (i=16; i<20; i++) {
            glTexCoord2f(data[5*i],data[5*i+1]);
            glVertex3f(data[5*i+2],data[5*i+3],data[5*i+4]);
        }
        glEnd();
}



// Calculates v=vM, M Is 4x4 In Column-Major, v Is 4dim. Row (i.e. "Transposed")
void VMatMult(GLfloat *M, GLfloat *v) {
    GLfloat res[3];
    res[0]=M[ 0]*v[0]+M[ 1]*v[1]+M[ 2]*v[2]+M[ 3]*v[3];
    res[1]=M[ 4]*v[0]+M[ 5]*v[1]+M[ 6]*v[2]+M[ 7]*v[3];
    res[2]=M[ 8]*v[0]+M[ 9]*v[1]+M[10]*v[2]+M[11]*v[3];;
    v[0]=res[0];
    v[1]=res[1];
    v[2]=res[2];
    v[3]=M[15];                                         // Homogenous Coordinate
}



void SetUpBumps(GLfloat *n, GLfloat *c, GLfloat *l, GLfloat *s, GLfloat *t) {
    GLfloat v[3];                           // Vertex From Current Position To Light
    GLfloat lenQ;                           // Used To Normalize

    // Calculate v From Current Vector c To Lightposition And Normalize v
    v[0]=l[0]-c[0];
    v[1]=l[1]-c[1];
    v[2]=l[2]-c[2];
    lenQ=(GLfloat) sqrt(v[0]*v[0]+v[1]*v[1]+v[2]*v[2]);
    v[0]/=lenQ;     v[1]/=lenQ;     v[2]/=lenQ;
    // Project v Such That We Get Two Values Along Each Texture-Coordinat Axis.
    c[0]=(s[0]*v[0]+s[1]*v[1]+s[2]*v[2])*MAX_EMBOSS;
    c[1]=(t[0]*v[0]+t[1]*v[1]+t[2]*v[2])*MAX_EMBOSS;
}


void doMesh1TexelUnits(void) {

    GLfloat c[4]={0.0,0.0,0.0,1.0};             // Holds Current Vertex
    GLfloat n[4]={0.0,0.0,0.0,1.0};             // Normalized Normal Of Current Surface
    GLfloat s[4]={0.0,0.0,0.0,1.0};             // s-Texture Coordinate Direction, Normalized
    GLfloat t[4]={0.0,0.0,0.0,1.0};             // t-Texture Coordinate Direction, Normalized
    GLfloat l[4];                                   // Holds Our Lightposition To Be Transform
ed Into Object Space
    GLfloat Minv[16];                               // Holds The Inverted Modelview Matrix To
Do So.
    int i;

    //glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);   // Clear The Screen And The Depth Buff
er
    // Build Inverse Modelview Matrix First. This Substitutes One Push/Pop With One glLoadIdentity
```

```
();
    // Simply Build It By Doing All Transformations Negated And In Reverse Order.

    glLoadIdentity();
    glRotatef(-yrot,0.0,1.0,0.0);
    glRotatef(-xrot,1.0,0.0,0.0);
    glTranslatef(0.0,0.0,-z);
//  doTable();
    glGetFloatv(GL_MODELVIEW_MATRIX,Minv);
    glLoadIdentity();
    glTranslatef(0.0,0.0,z);

    glRotatef(xrot,1.0,0.0,0.0);
    glRotatef(yrot,0.0,1.0,0.0);
//  doTable();
    // Transform The Lightposition Into Object Coordinates:
    l[0]=light_position[0];
    l[1]=light_position[1];
    l[2]=light_position[2];
    l[3]=1.0;                                           // Homogenous Coordinate
    VMatMult(Minv,l);

/*  PASS#1: Use Texture "Bump"
            No Blend
            No Lighting
            No Offset Texture-Coordinates */
    glBindTexture(GL_TEXTURE_2D, bump[filter]);
    glDisable(GL_BLEND);
    glDisable(GL_LIGHTING);
    doCube();
    doTable();
    doPedistal();
    drawstuff();
//  glLoadIdentity();
/* PASS#2:  Use Texture "Invbump"
            Blend GL_ONE To GL_ONE
            No Lighting
            Offset Texture Coordinates
            */
    glBindTexture(GL_TEXTURE_2D,invbump[filter]);
    glBlendFunc(GL_ONE,GL_ONE);
    glDepthFunc(GL_LEQUAL);
    glEnable(GL_BLEND);


    glBegin(GL_QUADS);
        // Back Face
        n[0]=0.0;       n[1]=0.0;       n[2]=1.0;
        s[0]=-1.0;      s[1]=0.0;       s[2]=0.0;
        t[0]=0.0;       t[1]=1.0;       t[2]=0.0;
        for (i=0; i<4; i++) {
            c[0]=data[5*i+2];
            c[1]=data[5*i+3];
            c[2]=data[5*i+4];
            SetUpBumps(n,c,l,s,t);
            glTexCoord2f(data[5*i]+c[0], data[5*i+1]+c[1]);
            glVertex3f(data[5*i+2], data[5*i+3], data[5*i+4]);
        }
        // Right Face
        n[0]=-1.0;      n[1]=0.0;       n[2]=0.0;
        s[0]=0.0;       s[1]=0.0;       s[2]=-1.0;
        t[0]=0.0;       t[1]=1.0;       t[2]=-1.0;
        for (i=4; i<8; i++) {
            c[0]=data[5*i+2];
            c[1]=data[5*i+3];
            c[2]=data[5*i+4];
            SetUpBumps(n,c,l,s,t);
            glTexCoord2f(data[5*i]+c[0], data[5*i+1]+c[1]);
            glVertex3f(data[5*i+2], data[5*i+3], data[5*i+4]);
        }
        // Left face
        n[0]=1.0;       n[1]=0.0;       n[2]=0.0;
        s[0]=0.0;       s[1]=0.0;       s[2]= 1.0;
```

```
        t[0]=0.0;       t[1]= -1.0;     t[2]=-1.0;
        for (i=8; i<12; i++) {
            c[0]=data[5*i+2];
            c[1]=data[5*i+3];
            c[2]=data[5*i+4];
            SetUpBumps(n,c,l,s,t);
            glTexCoord2f(data[5*i]+c[0], data[5*i+1]+c[1]);
            glVertex3f(data[5*i+2], data[5*i+3], data[5*i+4]);
        }
/*      // Right Face
        n[0]=-1.0;      n[1]=0.0;       n[2]=0.0;
        s[0]=0.0;       s[1]=0.0;       s[2]=-1.0;
        t[0]=0.0;       t[1]=1.0;       t[2]=0.0;
        for (i=12; i<16; i++) {
            c[0]=data[5*i+2];
            c[1]=data[5*i+3];
            c[2]=data[5*i+4];
            SetUpBumps(n,c,l,s,t);
            glTexCoord2f(data[5*i]+c[0], data[5*i+1]+c[1]);
            glVertex3f(data[5*i+2], data[5*i+3], data[5*i+4]);
        }
        // Left Face
        n[0]=1.0;       n[1]=0.0;       n[2]=0.0;
        s[0]=0.0;       s[1]=0.0;       s[2]=1.0;
        t[0]=0.0;       t[1]=1.0;       t[2]=0.0;
        for (i=16; i<20; i++) {
            c[0]=data[5*i+2];
            c[1]=data[5*i+3];
            c[2]=data[5*i+4];
            SetUpBumps(n,c,l,s,t);
            glTexCoord2f(data[5*i]+c[0], data[5*i+1]+c[1]);
            glVertex3f(data[5*i+2], data[5*i+3], data[5*i+4]);
        } */
/*      // Left Face
        n[0]=-1.0;      n[1]=0.0;       n[2]=0.0;
        s[0]=0.0;       s[1]=0.0;       s[2]=1.0;
        t[0]=0.0;       t[1]=1.0;       t[2]=0.0;
        for (i=20; i<24; i++) {
            c[0]=data[5*i+2];
            c[1]=data[5*i+3];
            c[2]=data[5*i+4];
            SetUpBumps(n,c,l,s,t);
            glTexCoord2f(data[5*i]+c[0], data[5*i+1]+c[1]);
            glVertex3f(data[5*i+2], data[5*i+3], data[5*i+4]);
        } */
    glEnd();

/* PASS#3:  Use Texture "Base"
            Blend GL_DST_COLOR To GL_SRC_COLOR (Multiplies By 2)
            Lighting Enabled
            No Offset Texture-Coordinates
            */
    if (!emboss) {
        glTexEnvf (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
        glBindTexture(GL_TEXTURE_2D,texture[filter]);
        glBlendFunc(GL_DST_COLOR,GL_SRC_COLOR);
        glEnable(GL_LIGHTING);
        doCube();
    }

    xrot+=xspeed;
    yrot+=yspeed;
    if (xrot>360.0) xrot-=360.0;
    if (xrot<0.0) xrot+=360.0;
    if (yrot>360.0) yrot-=360.0;
    if (yrot<0.0) yrot+=360.0;


                                // Keep Going
}


void doMeshNoBumps(void) {
```

```
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Clear The Screen And The Depth Buffer
    glLoadIdentity();                                   // Reset The View

    glRotatef(xrot,1.0,0.0,0.0);
    glRotatef(yrot,0.0,1.0,0.0);

        glActiveTextureARB(GL_TEXTURE1_ARB);
        glDisable(GL_TEXTURE_2D);
        glActiveTextureARB(GL_TEXTURE0_ARB);

    glDisable(GL_BLEND);
    glBindTexture(GL_TEXTURE_2D,texture[filter]);
    glBlendFunc(GL_DST_COLOR,GL_SRC_COLOR);
    glEnable(GL_LIGHTING);
    doCube();

    xrot+=xspeed;
    yrot+=yspeed;
    if (xrot>360.0) xrot-=360.0;
    if (xrot<0.0) xrot+=360.0;
    if (yrot>360.0) yrot-=360.0;
    if (yrot<0.0) yrot+=360.0;


                                // Keep Going
}

void doRoom(void) {

/*  glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    glColor3f(1.0, 1.0, 1.0);

    glPushMatrix();
    glRotatef(-yrot,0.0,1.0,0.0);
    glRotatef(-xrot,1.0,0.0,0.0);
    glTranslatef(0.0,0.0,-z); */
    glBindTexture(GL_TEXTURE_2D, tandb);
    glBegin(GL_QUADS);
        // Top Face
        glNormal3f( 0.0, -1.0, 0.0);
//      glColor3f(0.0, 1.0, 0.0);
        glTexCoord2f(0.0f, -1.0f); glVertex3f(-35.0f,  15.0f, -30.0f);  // Top Left Of The Texture
 and Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-35.0f,  15.0f,  30.0f);   // Bottom Left Of The Text
ure and Quad
        glTexCoord2f(-1.0f, 0.0f); glVertex3f( 35.0f,  15.0f,  30.0f);  // Bottom Right Of The Tex
ture and Quad
        glTexCoord2f(-1.0f, -1.0f); glVertex3f( 35.0f,  15.0f, -30.0f); // Top Right Of The Textur
e and Quad

        // Bottom Face
        glNormal3f( 0.0, 1.0, 0.0);
//      glColor3f(0.0, 0.0, 1.0);
        glTexCoord2f(-1.0f, -1.0f); glVertex3f(-35.0f, -15.0f, -30.0f); // Top Right Of The Textur
e and Quad
        glTexCoord2f(0.0f, -1.0f); glVertex3f( 35.0f, -15.0f, -30.0f);  // Top Left Of The Texture
 and Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 35.0f, -15.0f,  30.0f);   // Bottom Left Of The Text
ure and Quad
        glTexCoord2f(-1.0f, 0.0f); glVertex3f(-35.0f, -15.0f,  30.0f);  // Bottom Right Of The Tex
ture and Quad
    glEnd();


}



void doWalls(void) {

    glBegin(GL_QUADS);
        // Back Face
```

```
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-35.0f, -15.0f, -30.0f);   // Bottom Right Of The Tex
ture and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-35.0f,  15.0f, -30.0f);   // Top Right Of The Textur
e and Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f( 35.0f,  15.0f, -30.0f);   // Top Left Of The Texture
 and Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 35.0f, -15.0f, -30.0f);   // Bottom Left Of The Text
ure and Quad

        // Top Face

        glTexCoord2f(0.0f, 1.0f); glVertex3f(-35.0f,  15.0f, -30.0f);   // Top Left Of The Texture
 and Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-35.0f,  15.0f,  30.0f);   // Bottom Left Of The Text
ure and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f( 35.0f,  15.0f,  30.0f);   // Bottom Right Of The Tex
ture and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f( 35.0f,  15.0f, -30.0f);   // Top Right Of The Textur
e and Quad


        glTexCoord2f(1.0f, 1.0f); glVertex3f(-35.0f, -15.0f, -30.0f);   // Top Right Of The Textur
e and Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f( 35.0f, -15.0f, -30.0f);   // Top Left Of The Texture
 and Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 35.0f, -15.0f,  30.0f);   // Bottom Left Of The Text
ure and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-35.0f, -15.0f,  30.0f);   // Bottom Right Of The Tex
ture and Quad

        // Right face

        glTexCoord2f(1.0f, 0.0f); glVertex3f( 35.0f, -15.0f, -30.0f);   // Bottom Right Of The Tex
ture and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f( 35.0f,  15.0f, -30.0f);   // Top Right Of The Textur
e and Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f( 35.0f,  15.0f,  30.0f);   // Top Left Of The Texture
 and Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 35.0f, -15.0f,  30.0f);   // Bottom Left Of The Text
ure and Quad

        // Left Face

        glTexCoord2f(0.0f, 0.0f); glVertex3f(-35.0f, -15.0f, -30.0f);   // Bottom Left Of The Text
ure and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-35.0f, -15.0f,  30.0f);   // Bottom Right Of The Tex
ture and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-35.0f,  15.0f,  30.0f);   // Top Right Of The Textur
e and Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-35.0f,  15.0f, -30.0f);   // Top Left Of The Texture
 and Quad
    glEnd();


}

void doTable(void) {

    glBindTexture(GL_TEXTURE_2D, ped);
    glBegin(GL_QUADS);

        // Front Face
        glNormal3f(0.0, 0.0, 1.0);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-8.0f, -7.0f,  4.0f);  // Bottom Left Of The Texture
and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f( 8.0f, -7.0f,  4.0f);  // Bottom Right Of The Texture
 and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f( 8.0f,  -8.0f,  4.0f); // Top Right Of The Texture an
d Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-8.0f,  -8.0f,  4.0f); // Top Left Of The Texture and
 Quad
```

```
        //Back face
        glNormal3f(0.0, 0.0, -1.0);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-8.0f, -7.0f, -4.0f);  // Bottom Right Of The Texture
 and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-8.0f,  -8.0f, -4.0f); // Top Right Of The Texture an
d Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f( 8.0f,  -8.0f, -4.0f); // Top Left Of The Texture and
 Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 8.0f, -7.0f, -4.0f);  // Bottom Left Of The Texture
and Quad

        // Top Face
        glNormal3f(0.0, 1.0, 0.0);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-8.0f,  -7.0f, -4.0f); // Top Left Of The Texture and
 Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-8.0f,  -8.0f,  4.0f); // Bottom Left Of The Texture
and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f( 8.0f,  -8.0f,  4.0f); // Bottom Right Of The Texture
 and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f( 8.0f,  -7.0f, -4.0f); // Top Right Of The Texture an
d Quad

        // Bottom Face
        glNormal3f(0.0, -1.0, 0.0);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-8.0f, -7.0f, -4.0f);  // Top Right Of The Texture an
d Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f( 8.0f, -7.0f, -4.0f);  // Top Left Of The Texture and
 Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 8.0f, -8.0f,  4.0f);  // Bottom Left Of The Texture
and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-8.0f, -8.0f,  4.0f);  // Bottom Right Of The Texture
 and Quad

        // Right face
        glNormal3f(1.0, 0.0, 0.0);
        glTexCoord2f(1.0f, 0.0f); glVertex3f( 8.0f, -7.0f, -4.0f);  // Bottom Right Of The Texture
 and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f( 8.0f,  -8.0f, -4.0f); // Top Right Of The Texture an
d Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f( 8.0f,  -8.0f,  4.0f); // Top Left Of The Texture and
 Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 8.0f, -7.0f,  4.0f);  // Bottom Left Of The Texture
and Quad

        // Left Face
        glNormal3f(-1.0, 0.0, 0.0);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-8.0f, -7.0f, -4.0f);  // Bottom Left Of The Texture
and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-8.0f, -7.0f,  4.0f);  // Bottom Right Of The Texture
 and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-8.0f,  -8.0f,  4.0f); // Top Right Of The Texture an
d Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-8.0f,  -8.0f, -4.0f); // Top Left Of The Texture and
 Quad
    glEnd();

    glBegin(GL_QUADS);

        // Front Face
        glNormal3f(0.0, 0.0, 1.0);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-0.5f, -8.0f,  0.5f);  // Bottom Left Of The Texture
and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f( 0.5f, -8.0f,  0.5f);  // Bottom Right Of The Texture
 and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f( 0.5f,  -14.0f,  0.5f);    // Top Right Of The Textur
e and Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-0.5f,  -14.0f,  0.5f);    // Top Left Of The Texture
 and Quad

        //Back face
        glNormal3f(0.0, 0.0, -1.0);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-0.5f, -8.0f, -0.5f);  // Bottom Right Of The Texture
 and Quad
```

```
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-0.5f,  -15.0f, -0.5f);    // Top Right Of The Textur
e and Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f( 0.5f,  -15.0f, -0.5f);    // Top Left Of The Texture
 and Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 0.5f, -8.0f, -0.5f);  // Bottom Left Of The Texture
and Quad

        // Top Face
        glNormal3f(0.0, 1.0, 0.0);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-0.5f,  -8.0f, -0.5f); // Top Left Of The Texture and
 Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-0.5f,  -14.5f,  0.5f);    // Bottom Left Of The Text
ure and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f( 0.5f,  -14.5f,  0.5f);    // Bottom Right Of The Tex
ture and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f( 0.5f,  -8.0f, -0.5f); // Top Right Of The Texture an
d Quad

        // Bottom Face
        glNormal3f(0.0, -1.0, 0.0);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-0.5f, -8.0f, -0.5f);  // Top Right Of The Texture an
d Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f( 0.5f, -8.0f, -0.5f);  // Top Left Of The Texture and
 Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 0.5f, -14.5f,  0.5f); // Bottom Left Of The Texture
and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-0.5f, -14.5f,  0.5f); // Bottom Right Of The Texture
 and Quad

        // Right face
        glNormal3f(1.0, 0.0, 0.0);
        glTexCoord2f(1.0f, 0.0f); glVertex3f( 0.5f, -8.0f, -0.5f);  // Bottom Right Of The Texture
 and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f( 0.5f,  -14.5f, -0.5f);    // Top Right Of The Textur
e and Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f( 0.5f,  -14.5f,  0.5f);    // Top Left Of The Texture
 and Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 0.5f, -8.0f,  0.5f);  // Bottom Left Of The Texture
and Quad

        // Left Face
        glNormal3f(-1.0, 0.0, 0.0);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-0.5f, -8.0f, -0.5f);  // Bottom Left Of The Texture
and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-0.5f, -8.0f,  0.5f);  // Bottom Right Of The Texture
 and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-0.5f,  -14.5f,  0.5f);    // Top Right Of The Textur
e and Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-0.5f,  -14.5f, -0.5f);    // Top Left Of The Texture
 and Quad
    glEnd();

    glBegin(GL_QUADS);

        // Front Face
        glNormal3f(0.0, 0.0, 1.0);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-2.0f, -15.0f,  2.0f); // Bottom Left Of The Texture
and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f( 2.0f, -15.0f,  2.0f); // Bottom Right Of The Texture
 and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f( 2.0f,  -14.5f,  2.0f);    // Top Right Of The Textur
e and Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-2.0f,  -14.5f,  2.0f);    // Top Left Of The Texture
 and Quad

        //Back face
        glNormal3f(0.0, 0.0, -1.0);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-2.0f, -15.0f, -2.0f); // Bottom Right Of The Texture
 and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-2.0f,  -14.5f, -2.0f);    // Top Right Of The Textur
e and Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f( 2.0f,  -14.5f, -2.0f);    // Top Left Of The Texture
 and Quad
```

```
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 2.0f, -15.0f, -2.0f); // Bottom Left Of The Texture
and Quad

        // Top Face
        glNormal3f(0.0, 1.0, 0.0);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-2.0f,  -15.0f, -2.0f);    // Top Left Of The Texture
 and Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-2.0f,  -14.5f,  2.0f);    // Bottom Left Of The Text
ure and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f( 2.0f,  -14.5f,  2.0f);    // Bottom Right Of The Tex
ture and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f( 2.0f,  -15.0f, -2.0f);    // Top Right Of The Textur
e and Quad

        // Bottom Face
        glNormal3f(0.0, -1.0, 0.0);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-2.0f, -15.0f, -2.0f); // Top Right Of The Texture an
d Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f( 2.0f, -15.0f, -2.0f); // Top Left Of The Texture and
 Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 2.0f, -14.5f,  2.0f); // Bottom Left Of The Texture
and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-2.0f, -14.5f,  2.0f); // Bottom Right Of The Texture
 and Quad

        // Right face
        glNormal3f(1.0, 0.0, 0.0);
        glTexCoord2f(1.0f, 0.0f); glVertex3f( 2.0f, -15.0f, -2.0f); // Bottom Right Of The Texture
 and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f( 2.0f,  -14.5f, -2.0f);    // Top Right Of The Textur
e and Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f( 2.0f,  -14.5f,  2.0f);    // Top Left Of The Texture
 and Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 2.0f, -15.0f,  2.0f); // Bottom Left Of The Texture
and Quad

        // Left Face
        glNormal3f(-1.0, 0.0, 0.0);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-2.0f, -15.0f, -2.0f); // Bottom Left Of The Texture
and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-2.0f, -15.0f,  2.0f); // Bottom Right Of The Texture
 and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-2.0f,  -14.5f,  2.0f);    // Top Right Of The Textur
e and Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-2.0f,  -14.5f, -2.0f);    // Top Left Of The Texture
 and Quad
    glEnd();

}

void BuildList(void)
{
    int i, j, k, m;
    double ptr[3];
    glShadeModel(GL_SMOOTH);
    dlists = glGenLists(4);
    for(m = 0; m < numobjs; m++)
    {

    glNewList(dlists + m, GL_COMPILE);
        moveone(m);
        for(i = 0; i < object[m].faces; i++)
        {
            glBegin(GL_POLYGON);

            for(j = 1; j <= object[m].f_list[i][0]; j++)
            {
                for(k=0;k<3;k++)
                    ptr[k]=-1.0 * object[m].nrmls[(object[m].f_list[i][j])][k];
                    glNormal3dv(ptr);
                    glVertex3dv(object[m].v_list[(object[m].f_list[i][j])]);
            }
            glEnd();
```

```
        }
        revmoveone(m);
    glEndList();
    }

}


void drawstuff(void)
{
    int i;


    glGetFloatv(GL_MODELVIEW_MATRIX,Orig);
    glEnable(GL_TEXTURE_GEN_S);
    glEnable(GL_TEXTURE_GEN_T);
    glBindTexture(GL_TEXTURE_2D, tabl);

    glDisable(GL_TEXTURE_GEN_S);
    glDisable(GL_TEXTURE_GEN_T);

    glEnable(GL_TEXTURE_GEN_S);
    glEnable(GL_TEXTURE_GEN_T);
    glBindTexture(GL_TEXTURE_2D, tobj[0]);
    glCallList(dlists);
    glDisable(GL_TEXTURE_GEN_S);
    glDisable(GL_TEXTURE_GEN_T);
    if(sflag)
    {
        glEnable(GL_TEXTURE_GEN_S);
        glEnable(GL_TEXTURE_GEN_T);
        glBindTexture(GL_TEXTURE_2D, tobj[4]);
        glCallList(dlists + 4);
        glDisable(GL_TEXTURE_GEN_S);
        glDisable(GL_TEXTURE_GEN_T);
    }
    if(wflag)
    {
        glEnable(GL_TEXTURE_GEN_S);
        glEnable(GL_TEXTURE_GEN_T);
        glBindTexture(GL_TEXTURE_2D, tobj[5]);
        glCallList(dlists + 5);
        glDisable(GL_TEXTURE_GEN_S);
        glDisable(GL_TEXTURE_GEN_T);
    }
    if(aflag)
    {
        glTranslatef(0.0, 2.0, 0.0);
        glTranslatef(1.0, 0.0, 0.0);
//      if( (rt%2) == 0)
            glRotatef(mv, 0.0, 1.0, 0.0);
//      else
//          glTranslatef(0.3, 0.0, 0.0);
    }

        for(i = 1; i < numobjs - 2;  i++)
    {
        glEnable(GL_TEXTURE_GEN_S);
        glEnable(GL_TEXTURE_GEN_T);
        glBindTexture(GL_TEXTURE_2D, tobj[i]);
        glCallList(dlists + i);
        glDisable(GL_TEXTURE_GEN_S);
        glDisable(GL_TEXTURE_GEN_T);
    }
    if(aflag)
    {
//      if( (rt%2) == 0)
            glRotatef(-mv, 0.0, 1.0, 0.0);
//      else
//          glTranslatef(-0.3, 0.0, 0.0);
        glTranslatef(-1.0, 0.0, 0.0);
        glTranslatef(0.0, -2.0, 0.0);
    }
```

```
    glPopMatrix();
    glLoadIdentity();
    glLoadMatrixf(Orig);
}


void moveone(int q)
{
    if(q == 0)
    {
        glRotatef(90.0, 0.0, 1.0, 0.0);
        glRotatef(90.0, -1.0, 0.0, 0.0);
        glTranslatef(0.0, 0.0, -6.0);
        glTranslatef(0.0, 10.0, 0.0);
    }
    else if (q == 1)
        glTranslatef(0.0, -7.0, 0.0);
    else if (q == 2)
    {
        glTranslatef(0.0, -5.5, 0.0);
        glTranslatef(-1.0, 0.0, 0.0);
    }
    else if(q == 3)
    {
        glRotatef(90.0, 0.0, -1.0, 0.0);
        glTranslatef(0.0, 0.0, -30.0);
        glTranslatef(0.0, 2.8, 0.0);
        glTranslatef(-26.0, 0.0, 0.0);
    }
    else if(q == 4)
    {
        glRotatef(90.0, -1.0, 0.0, 0.0);
        glTranslatef(0.0, 10.0, 0.0);
    }
    else if(q == 5)
    {
        glRotatef(90.0, -1.0, 0.0, 0.0);
        glRotatef(90.0, 0.0, 0.0, 1.0);
        glTranslatef(10.0, 0.0, 0.0);
        glTranslatef(0.0, 0.0, -2.0);
        glTranslatef(-5.0, 0.0, 0.0);
        glScalef(1.0, 1.0, 1.08);
    }
}

void revmoveone(int q)
{
    if(q == 0)
    {
        glTranslatef(0.0, -10.0, 0.0);
        glTranslatef(0.0, 0.0, 6.0);
        glRotatef(90.0, 1.0, 0.0, 0.0);
        glRotatef(90.0, 0.0, -1.0, 0.0);
    }
    else if(q == 1)
        glTranslatef(0.0, 7.0, 0.0);
    else if (q == 2)
    {
        glTranslatef(1.0, 0.0, 0.0);
        glTranslatef(0.0, 5.5, 0.0);

    }
    else if(q == 3)
    {
        glTranslatef(26.0, 0.0, 0.0);
        glTranslatef(0.0, -2.8, 0.0);
        glTranslatef(0.0, 0.0, 30.0);
        glRotatef(90.0, 0.0, 1.0, 0.0);
    }
    else if(q == 4)
    {
        glRotatef(90.0, 1.0, 0.0, 0.0);
        glTranslatef(0.0, 10.0, 0.0);
```

```
    }
    else if(q == 5)
    {
        glRotatef(90.0, 1.0, 0.0, 0.0);
        glRotatef(90.0, 0.0, 0.0, -1.0);
        glTranslatef(-10.0, 0.0, 0.0);
        glTranslatef(0.0, 0.0, 2.0);
        glTranslatef(5.0, 0.0, 0.0);
        glScalef(1.0, 1.0, -1.08);
    }
}

void doPedistal(void)
{
    glBindTexture(GL_TEXTURE_2D, tabl);
    glBegin(GL_QUADS);

        // Front Face
        glNormal3f(0.0, 0.0, 1.0);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(30.0f, -14.9f, -25.0f);    // Bottom Left Of The Text
ure and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f( 34.0f, -14.9f, -25.0f);   // Bottom Right Of The Tex
ture and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f( 34.0f, 0.0f, -25.0f); // Top Right Of The Texture an
d Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f(30.0f, 0.0f, -25.0f);  // Top Left Of The Texture and
 Quad

        //Back face
        glNormal3f(0.0, 0.0, -1.0);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(30.0f, -14.9f, -29.0f);    // Bottom Right Of The Tex
ture and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f(30.0f, 0.0f, -29.0f);  // Top Right Of The Texture an
d Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f( 34.0f, 0.0f, -29.0f); // Top Left Of The Texture and
 Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 34.0f, -14.9f, -29.0f);   // Bottom Left Of The Text
ure and Quad

        // Top Face
        glNormal3f(0.0, 1.0, 0.0);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(30.0f,  0.0f, -29.0f); // Top Left Of The Texture and
 Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f(30.0f,  0.0f, -25.0f); // Bottom Left Of The Texture
and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f(34.0f,  0.0f, -25.0f); // Bottom Right Of The Texture
 and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f(34.0f,  0.0f, -29.0f); // Top Right Of The Texture an
d Quad

        // Bottom Face
        glNormal3f(0.0, -1.0, 0.0);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(34.0f, -14.9f, -29.0f);    // Top Right Of The Textur
e and Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f(30.0f, -14.9f, -29.0f);    // Top Left Of The Texture
 and Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f(30.0f, -14.9f, -25.0f);    // Bottom Left Of The Text
ure and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f(34.0f, -14.9f,  -25.0f);   // Bottom Right Of The Tex
ture and Quad

        // Right face
        glNormal3f(1.0, 0.0, 0.0);
        glTexCoord2f(1.0f, 0.0f); glVertex3f( 34.0f, -14.9f, -29.0f);   // Bottom Right Of The Tex
ture and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f( 34.0f,  0.0f, -29.0f);    // Top Right Of The Textur
e and Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f( 34.0f,  0.0f,  -25.0f);   // Top Left Of The Texture
 and Quad
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 34.0f, -14.9f, -25.0f);   // Bottom Left Of The Text
ure and Quad

        // Left Face
```

```
        glNormal3f(-1.0, 0.0, 0.0);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(30.0f, -14.9f, -29.0f);    // Bottom Left Of The Text
ure and Quad
        glTexCoord2f(1.0f, 0.0f); glVertex3f(30.0f, -14.9f, -25.0f);    // Bottom Right Of The Tex
ture and Quad
        glTexCoord2f(1.0f, 1.0f); glVertex3f(30.0f,  0.0f, -25.0f); // Top Right Of The Texture an
d Quad
        glTexCoord2f(0.0f, 1.0f); glVertex3f(30.0f,  0.0f, -29.0f); // Top Left Of The Texture and
 Quad
    glEnd();
}

void animate (void)
{
    int i, mult;

//    rt = rand();
      i = 0;
      mult = mv/360;
      i = mult * 360;
      mv = mv + 1;
      if((mv > 50 + i) && (mv < 70 + i))
          wflag = 1;
      else
        wflag = 0;

      if((mv > 60 + i) && (mv < 80 + i))
        sflag = 1;
      else
        sflag = 0;
}
```