Holography and Kinect

Tom Groentjes, Leiden University

January 31, 2013

Albert Einstein:

“Reality is merely an illusion, albeit a very persistent one.”
Abstract

Holographic images are often presented as static images. This project aims at solutions to depict a dynamic ‘holographic’ image that can be manipulated. First an illusion is created of a generic object floating in mid air by using a Mirascope, a mirror room that consists of two concave parabolic mirrors stacked on top of each other. The top mirror has an aperture in the center to allow viewing inside of the mirror room. The generic object that is placed inside the mirascope then appears to be floating in mid-air just above the top mirror. The object can be made dynamic by projecting images onto it, using 2 beamers (one beaming onto the frontal surface of the object and one beaming onto the backside of the object). Subsequently the Kinect, an input device developed by Microsoft, that recognizes movement and gestures, is used to manipulate the image beamed onto the object. Gestures will change the projections beamed onto the object, and therefore changing the illusion of the floating object above the mirascope.

Keywords

Holography, Mirascope, Illusion, Spherical Mirrors, Microsoft Kinect, Interface, Simulation, Optics, OpenGL, GLUT, OpenNI, Libfreenect, Globe, Sphere, human binocular disparity, parallax barrier, anaglyph, gesture recognition
1 Introduction

There exists a wide variety of methods to create a hologram, where only some of them are dynamic, thus far there have not been holograms that could be manipulated by gestures onto or around the hologram. The Kinect, an input device developed by Microsoft, that recognizes movement and gestures, is used to manipulate the image beamed onto the object. Gestures will change the projections beamed onto the object, and therefore changing the illusion of the floating object above the mirascope. This paper is organized as follows, first in Section 2 current techniques and state of the art technologies will be discussed, then in Section 3 the materials and models including theory of the mirascope, anamorphoses, and kinect input are discussed. Then in Section 4 the implemented functionality will be explained. Some experiments are being discussed in Section 5. The experimented results being discussed in Section 6. The possibilities for conclusions can be found in Section 7. Finally the paper ends discussing future research in Section 8.

2 State of the art

A picture made by holography, which has its etymological roots in the Greek words for ‘whole’ and ‘drawing’, is called a hologram. A hologram as such is a complete drawing that, when viewed from different angles, shows the respective sides of the original object. The recording method resembles the way that music is recorded. The vibrations that are caused by music at a specific place over time, can be recorded in such a way that the original music can be reconstructed, even if the source is not present anymore. When sound is recorded the waves of sound pressure are translated into waves of electrical voltage. These waves of electrical voltage are then turned into magnetism (on tape) or grooves (on a record). The magnetism on tapes and the waves on a record can be translated back into electrical voltage, that in turn can be translated into waves to reproduce the original sound.

There are multiple ways to make holograms, fixed or dynamic. The best known is the traditional hologram which is made using a laser and a recording medium (often a silver mirror). Much like a photo is imprinted on the photographic film, a hologram is imprinted on the recording medium, with the distinction that a direct light source to the medium is needed (the reference beam, Figure 1 illustrates a possible setup). A few drawbacks of the
traditional technique are:

- that it creates a fixed interference image that can not be altered afterwards
- that it does not create a 360° image of the object but one that is limited to the part of the object that has light shown upon it.
- that the recording medium can not be reused
- that the slightest vibrations in the medium or light source can disrupt a good recording.

A lot of research has been done to create the illusion of a hologram or the illusion of a three dimensional image (3D) by using the human binocular disparity (HBD) [14, 15]. All these solutions take advantage of the fact that our brain helps us perceive the outside world, and constructs one image from the input it gets from two eyes. This feature of the human mind lets us see depth and assists in the perception of movement and direction. Three methods related to the exploitation of HBD and the human visual system are discussed, the anaglyph method, the polarization method and the parallax barrier method.

In the anaglyph method [23] the viewer wears a basic pair of glasses/filters
wherein one of the lenses is colored red and the other is colored cyan or green. The image that is viewed consists of 2 images that are placed on top of each other (they are slightly shifted, see Figure 2a), and both images are perceived by the viewer. One of the images will be seen by the left eye, and the other will be seen by the right eye (see Figure 2b). The brain then constructs a mental three dimensional image of the picture.

A similar approach that takes advantage of the HBD is the polarized 3D glasses approach. Instead of a red filter and a cyan filter the filters are both polarized. By choosing the polarization of the glasses in such a way that light that is destined to go to the left eye will pass through the left filter and light that is destined for the right eye, will pass through the right filter. This will give the same effect as the anaglyph method. This is realized by using a 90° difference between the polarization of the left and right filters.

![Anaglyph images](image)

Figure 2: Anaglyph images

Then there is the 3DS\textsuperscript{TM} technique of Nintendo\textsuperscript{®} that alternates images in high speed through a parallax barrier (see Figure 4b) to give the illusion of 3D [21].

A glass pyramid with projections on all sides gives the illusion of an object inside the pyramid like the viZoo\textsuperscript{TM} Cheoptics hologram [19] (see Figure 3a). An image is projected on top of the pyramid such that every side of the pyramid has a part of the object shown onto it. These images give the illusion of a 3 dimensional object inside of the pyramid.

There are computer inserted holograms [22] (see Figure 3b) in which a hologram is added to a picture after the picture has been taken. One or more images are added to a base image, such that it ‘looks’ like the added images are holograms in the base image.
A new plasma technology is being researched in Japan in which a focused laser is used to create plasma dots in mid air. By making multiple dots in rapid succession 3D-structures can be created in mid-air [24](see Figure 4a).

A life size 3D hologram pod is being researched at Queens University. In this pod a 2D-image is projected in relation to the position of the viewer, which allows the user to walk around the person that is displayed in the pod [25, 26]. The image inside the pod updates according to the position of the viewer, anamorphic techniques are used to display the image inside the pod in ‘normal’ dimensions. A slightly different implementation of a hologram is done by a mirascope, which by the use of mirrors, creates a phantom image of an original physical object [1]. The mirascope will be discussed further in section 3.

Creating a dynamic image is done by creating a projection of the image onto a solid surface. This projected image can be altered by changing the
projection. Multiple projection techniques exist. There are multiple projects in which anamorphic images are beamed onto a surface to make them appear in normal dimensions. P. Bourke created a dome that can show the hemisphere using one beamer and a spherical mirror [3].

A dynamic image can change on its own in a predefined fixed way, or it can change on input from a user. There are various ways for a user to dynamically manipulate an image, for example the Globe4D project [4], where a globe can be manipulated using basic keyboard commands and mouse scroll movements. There are however some alternatives to the traditional input devices. The WiiMote, Playstation® Move and the Kinect™ sensor, for example, offer new interaction methods such as gestures and movement.

3 Materials and methods

A possible solution to create a dynamic interfaceable hologram is based on the concept of an optical illusion with a red ball in a mirascope. The red ball in the illusion appears to be on top of a mirror, while in fact it is not. For a simplified picture of the concept see Figure 5. The mirascope uses a double parabolic mirror to create an image in space.

![Figure 5: Concept optical illusion using a mirascope](image)

The apparatus can be used to create a dynamic optical illusion by using the ball to show projected images from several directions. The projected image comes from two different projectors that beam a projection on to circular mirrors, which reflect the image into the mirascope onto a reflective object (in this case a white styrofoam ball). For a test setup see Figure 6.
3.1 Mirascope

The theory of a Mirascope (trademarked as Mirage® [7]) is partly explained by Sriya Adhya and John Noé [5]. The mirascope creates an optical illusion which makes a projection of an object appear while it is not there. This illusion is achieved by using two opposed concave parabolic mirrors which are placed on top of each other to create a sort of ‘mirror-room’. The top mirror has an aperture in which one can see the ‘floating’ object. The mirrors are concaved in such a way that the reflection of the image floats above the actual image as can be seen in Figure 5.

3.1.1 Mirascope math

The diameters of the mirascopes available vary, and so do its aperture and height. In Table 1 are a couple of sizes are shown and in Figure 7 and Figure 8 they are plotted. The parabolic function is derived using $F(x) = ax^2 + bx + c$ and filling in the $x$ and $F(x)$ values from height and width, where width is divided by 2 so the graphs fall nicely on top of the origin. The height is divided by 2 because we are only concerned about the bottom half of the mirascope.

The second derivative is mentioned because that is the height of the focal point of the mirror. As can be seen the focal point lies inside of the mirror.
Table 1: Mirror sizes

<table>
<thead>
<tr>
<th>Diameter</th>
<th>Height</th>
<th>Aperture</th>
<th>$F(x)$</th>
<th>second derivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>11</td>
<td>16</td>
<td>$F(x) = 0.010x^2$</td>
<td>$F''(x) = 0.02$</td>
</tr>
<tr>
<td>55.88</td>
<td>10.16</td>
<td>15.24</td>
<td>$F(x) = 0.0133x^2$</td>
<td>$F''(x) = 0.0266$</td>
</tr>
<tr>
<td>22.86</td>
<td>3.31</td>
<td>5.08</td>
<td>$F(x) = 0.0292x^2$</td>
<td>$F''(x) = 0.0584$</td>
</tr>
<tr>
<td>13.8</td>
<td>2.4</td>
<td>4.1</td>
<td>$F(x) = 0.053x^2$</td>
<td>$F''(x) = 0.106$</td>
</tr>
</tbody>
</table>

If those values are plotted in a single figure one can see the following.

![Figure 7: Graph showing different sizes of mirascopes](image1)

![Figure 8: Graph showing apertures of different sizes of mirascopes](image2)

We can deduce from Figure 7 that there is a clear relation between the $a$ in $F(x) = ax^2 + bx + c$ and the diameter of the mirascope. This relation is linear as can be seen in Figure 9. Simplified, that is the relation between the focal point of the mirror and the diameter.
Figure 9: Graph showing relation between diameter/2 and factor $a$ in $F(x) = ax^2 + bx + c$

Table 2: Aperture height relation

<table>
<thead>
<tr>
<th>Total height</th>
<th>Aperture</th>
<th>Aperture/Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>16</td>
<td>0.7272</td>
</tr>
<tr>
<td>20.32</td>
<td>15.24</td>
<td>0.75</td>
</tr>
<tr>
<td>6.62</td>
<td>5.08</td>
<td>0.7673</td>
</tr>
<tr>
<td>4.8</td>
<td>4.1</td>
<td>0.8542</td>
</tr>
</tbody>
</table>

We can further deduce from Table 2, that the aperture is around 0.75 times the total height (height of two mirrors combined).

3.1.2 Mirascope projection

The projection appears in the focal point of the top mirror. Therefore it appears inside the mirascope. The aperture makes it possible to see the image. The image can occur [5] at multiple positions, depending on the distance between the bottom and top mirror. However the image fades in relation to the distance of the two mirrors. The bigger the distance, the vaguer the image.

3.2 Image projection and anamorphosis

Because all images are projected onto an object (in this case a ball), it is necessary to preprocess the images to make them fit the object and take into account the curvature of the object. With the map of the world that is not necessary because the world is more or less a round shape. With other images however the images have to be stretched and shrunk so that their dimensions appear normal when projected on a ball. Anamorphosis is the technique used to achieve this (see Figure 10). More on anamorphoses can be found in a report by E. Vreeswijk [9].
3.3 Interaction Kinect

Microsoft’s Kinect, an input device that can register and recognize gestures and movements, offers the possibility to use gestures and movement to interact with a computer. For this project the Kinect is used as an input device to manipulate the movement of the projected image, here a globe.

3.4 Implementation

The OpenKinect-libfreenect library is used for interfacing with the Kinect [16]. Furthermore OpenGL (GLUT) is used for the 3D visualization and anamorphic projections of a globe textured with images resembling the planets in our solar system. The textures used, are depicted in Figures 11 and 12. The code can be found in Appendix A.

3.4.1 Tracking movement

The OpenKinect-libfreenect library provides the depth map that is generated by the Kinect sensor. This information is used to track the movement of the hand of the user. To get a ‘window’ in which movements will be tracked, all info that is not within 50 cm to 60 cm of the Kinect is deleted. That window is saved in a two-way array, where every cell in the array represents a pixel. If that pixel is 0, there is no object in that pixel. If that pixel is 1, it has an object. The center of the object in that frame is calculated by generating a smaller depth map from the existing depth map. The depth map from the Kinect has a resolution of 640x480 pixels and the code resizes
that in multiple steps to 20x15. In every step the size is four times smaller. Only pixels having at least two underlying pixels having an object are labeled as having an object. All other pixels are set to not having an object. After a series of steps a 20x15 pixels map with the location information of the object is available. Then the center of the object is calculated by checking in which column and in which row of the two-way array most objects are present. Those coordinates are used to track the actual movement of the hand/object. In every new frame the coordinates of the object are calculated, in such a way that in every frame the difference between the coordinates can be calculated. These values will be used to calculate the traveled distance of the object/hand, which can then be used to alter the projection. The altered projection will then reflect the movement. This implementation is used because the OpenKinect-libfreenect library, does not support skeletal recognition. Since the release of this library multiple other libraries have been released, like the Microsoft Kinect SDK which support a wide array of extra functions including skeletal tracking [20].

### 3.4.2 Implemented moves

If a hand is held in the center of the window for a couple of seconds, then the program starts calculating x and y coordinates for the hand. If the hand is moved to the left the globe will start turning left, and if the hand moves to the right the globe will turn to the right. For up and down similar moves are implemented. If the hand is removed from the window the globe stops spinning. If then the hand is reinserted into the window for a couple of seconds the program restarts calculating x and y coordinates. If ‘home’ is pressed the globe will keep spinning after the hand is removed from the window.

### 4 Functionality

**Default**

If ‘end’ is pressed the program will go to its initial configuration.

**End:** return to initial configuration
**Globe**

The projection is restricted to a sphere, because of the scope of this project, but future projects could change the object on which the projection is rendered, or can use the earlier discussed technique of anamorphosis [9] to project different kinds of objects on the sphere. For this project multiple images of planets of our solar system are used as textures. The feature to cycle through the planets is enabled by default, but can be disabled/enabled by pressing ‘c’ To cycle through the available textures the ‘+’ and ‘-’ keys on the numpad are used.

+ : Go to next planet

− : Go to previous planet

Del : Automatically switch planets

**Overlay**

To display more information onto the globe than just the texture, overlays are implemented. Overlays can be activated by pressing ‘t’ for enabling transparency. If overlays are set to *enabled* and the ‘home’ key is pressed, the overlays will animate on top of the sphere. If ‘home’ is pressed again the animation will stop. The overlay texture is a black vertical line on a white image that is made transparent to allow viewing of the underlying texture.

  t : Enable/Disable overlays

**Home** : Animate/Static overlays

**Kinect**

To accommodate for different setups, some interaction methods for the Kinect are implemented. When pressing ‘w’ the Kinect angle will go up, pressing ‘s’ makes the kinect angle go down, pressing ‘x’ resets the angle to default. The ‘|’ key changes the led light color of the kinect led light. If the DEBUG flag in the defines.h file is set to 1 the kinect feed can be changed between 8Bit, RGB and YUV_RGB by pressing ‘f’.

  w : Kinect angle ++
5 Experiments

5.1 Mirrors

There are multiple ways to produce the parabolic mirrors required for a mirascope. Retail mirrors with an internal focal point, and a diameter big enough to allow for beamers to project into the mirascope, are too expensive. There are relatively cheap mirrors available but they all have a focal point outside of the mirror. On greenpowerscience.com [10] they show a relatively simple way to create a parabolic mirror using silver ‘stickering’. On the website of experimenten.nl[11] they show a chemical way of making a mirror, and with a chrome spray[12] one can spray a mirror onto any surface. A parabolic shape can be created by rotating a bucket with water and plaster at a specific speed until the plaster and water create a parabolic shape, as discussed by Nogueira, Raggio and Koch Torres Assis[13]. If a plexiglas mirror is available it is possible to melt that into a mold just like on greenpowerscience.com[10]. At the website of Angelgilding.com [17] there is a drip silver mirroring kit available, which could also be used to create a mirrored surface onto a parabolic surface.

5.1.1 Do it yourself mirror procedure

To create mirrors of a big enough size to allow beamers to project into them, a plaster mold is needed. This because with a plaster mold the curvature can be perfected using sanding paper and other tools. A plexiglas mirror looks likes the best way to create a parabolic mirror. This mirror will be heated to above 250° such that it will sink into the mold (plexiglas becomes flexible at 250°). The plaster mold will be created using a strong, flexible, slightly stretchable foil (painters foil). The foil will be fixed to a wooden base that has pillars attached to it which combined agree with the dimensions needed.
The dimensions needed are diameter 65 cm, and height 11 cm. Then water is used to make the foil take a parabolic shape (fluid dynamics will make it take the shape of a parabola). Plaster will be added to the water to get a negative mold for the mirror. That mold will be used to create a counter mold (the positive mold) in which a plexiglas mirror will be melted using either an oven (if one can be found that is big enough to fit the molds), or a heating gun to heat the mirror to between 250° and 270° degrees.

5.2 Interactive anamorphic 3D projections

The interactive anamorphic 3D projection created using the programming language C, and OpenGL. The initial idea was to create a simulation using two beamers, however during the final demonstration some problems were encountered, mainly as a result of the fact that two different beamers with two different resolutions and two different cable lengths were used. To show a working demo on two screens the simulation has been changed, now using two different computer monitors. The base simulation is changed in such a way that in stead of providing the projections of both sides of the object, it will show only a quarter of the object, such that when shown on two different windows they will show half of the object. The interaction between the windows stays the same. This way the simulation will resemble the viZoo approach[19]. The simulation spans two different windows, each showing a part of the globe. One window showing the left side, and one showing the right side. When placed on two different monitors they will create a view of half of a complete globe.

5.3 Kinect manipulation

The Kinect interface as built for this project allows for manipulation of the simulation. If a hand is put in the center of the detection area, then the program starts recognizing gestures inputted by the user. If the hand is ‘swiped’ to the right the globe starts spinning to the right. This also works for swiping left, swiping up and swiping down.
6 Results

6.1 Mirrors
At this time, I was unable to create or acquire the mirrors needed for this project. As complete mirrors available in various online stores that fit our specifications greatly exceeded our budget. We attempted to create a mirror following the procedure described in Section 5.1.1. We succeeded in creating a plaster mold and attempted to heat a plexiglas mirror with a point heater to fit the shape of the mold. However, a point heater proved unsuitable because it cannot uniformly heat the plexiglas. From this experiment we learned that the best results will be achieved by using a large oven, as it allows for the necessary consistency of temperature (250°). Regretfully, we could not acquire the usage of such equipment during this project.

6.2 Simulation
The simulation created by the programming code in Appendix A shows a working interaction between two different OpenGL windows. Together they show half of a globe that can spin in multiple directions. The simulation provides some sort of illusion of a holographic sphere behind the screens if the screens are placed together in a convex fashion.

6.3 Kinect manipulation
Kinect input is registered and it is possible to manipulate the simulation using gestures. All implemented gestures work, however the input isn’t robust, and rather error prone. This can be accredited to the fact that the interaction has to be done in a predefined space.

7 Conclusion
The provided model/computer simulation suggests that if the mirascope is large enough, a holographic image can be created that can in turn be manipulated by hand-gestures that are recognized by the Kinect sensor.
8 Future work

For future work there are many possible ways to go. At this time the projection on the object is the element that varies, but a rotating object can be inserted into the mirascope to allow movement outside of the projection area (the center of the mirascope). Anamorphy can be implemented to generate projections onto other objects than globes on the spherical object. The pointclouds library [18] could be implemented to make the program more robust. The newer Kinect for windows SDK could be implemented to create a better interface with the system. Interaction possibilities could be expanded to accompany better interaction and a greater range of uses for the project.
References


[7] “Mirage” is a registered trademark of its manufacturer, Opti-Gone International, Ojai, CA 93023 USA. http://www.optigone.com


[22] Computer added animation / hologram:


[24] True 3D display Aerial Burton:


A Appendix A

A.1 glview.c

Programs Main function for glview.

```c
#include <stdio.h>
#include <thread.h>
#include "libfreenect.h"
#include "threadfunc.h"

int main(int argc, char **argv)
{
    program = GLORBS;
    FillArrayImageNames();
    // set frame in which to recognize gestures
    kinect_set_active_area(0,480,640,550,650);
    update = 0;
    meanxoud = 0;
    meanyoud = 0;
    int res;
    depth_mid = (uint8*)malloc(640*480*3);
    depth_front = (uint8*)malloc(640*480*3);
    rgb_back = (uint8*)malloc(640*480*3);
    rgb_mid = (uint8*)malloc(640*480*3);
    rgb_front = (uint8*)malloc(640*480*3);
    printf("Kinect\camera\test\n");

    int i;
    for (i=0; i<2048; i++)
    {
        float v = i/2048.0;
        v = powf(v, 3) * 6;
        int gamma[i] = v * 6 + 256;
    }

    if (freenect_init(&ctx, NULL) < 0) {
        printf("freenect\init\failed\n");
        return 1;
    }
```


### A.2 threadfunc.c

Main program to allow for multiple windows to interact with each other.

```c
#include <assert.h>
#include <stdio.h>
#include <GL/glut.h>
#include <GL/glu.h>
#include <pthread.h>
#include "libfreenect.h"
#include "threadfunc.h"

pthread_mutex_t g_backbuf_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t g_frame_cond = PTHREAD_COND_INITIALIZER;

freenect_video_format requested_format = FREENECT_VIDEO_RGB;

int got_rgb = 0;
int got_depth = 0;

void *g_threadfunc(void *arg){
    glutInit(&g_argc, g_argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(0, 0);
    /* Open a window */
    window = glutCreateWindow("2");  // first create the reacting window.
    /* Register the function to do all our OpenGL drawing. */
    glutDisplayFunc(&DrawGLScene2);
    /* Even if there are no events, redraw our gl scene. */
    glutIdleFunc(&DrawGLScene2);
    /* Register the function called when our window is resized. */
    glutReshapeFunc(&ReSizeGLScene2);
    /* Register the function called when the keyboard is pressed. */
    glutKeyboardFunc(&keyPressed);
    /* Register the function called when mouse left is clicked */
    glutMouseFunc(&MouseEvent);
    return 0;
}
```

```c
freenect_set_log_level(f_ctx, FREENECP_LOG_DEBUG);

int nr_devices = freenect_num_devices(f_ctx);
printf( "%d devices found \n", nr_devices);

int user_device_number = 0;
if (argc > 1) user_device_number = atoi(argv[1]);
if (nr_devices < 1) return 1;

int n_devices = freenect_num_devices(f_ctx);
printf("Number of devices found: %d\n", n_devices);

if (user_device_number < 0) return 1;

int use_device_number = 0;
if (argc > 1) use_device_number = atoi(argv[1]);
if (n_devices < 1) return 1;

if (freenect_open_device(f_ctx, &fdev, user_device_number) < 0) {
    printf("Could not open device\n");
    return 1;
}

res = pthread_create(&freenect_thread, NULL, freenect_threadfunc, NULL);
if (res) {
    printf("pthread_create failed\n");
    return 1;
}

pthreadfunc(NULL);
return 0;
```

// OS X requires GLUT to run on the main thread
```
/* Register the function called when special keys (arrows, page down, etc) are pressed */
glutSpecialFunc(&specialKeyPressed);
InitGL2(640, 480);
/* the window starts at the upper left corner of the screen */
glutInitWindowPosition(640, 0);
/* Open a second window */
window2 = glutCreateWindow("1");
/* Even if there are no events, redraw our gl scene. */
glutDisplayFunc(&DrawGLScene2);
/* Even if there are no events, redraw our gl scene. */
glutIdleFunc(&DrawGLScene2);
/* Register the function called when special keys (arrows, page down, etc) are pressed */
glutSpecialFunc(&specialKeyPressed);
InitGL2(640, 480);
/* Even if there are no events, redraw our gl scene. */
glutDisplayFunc(&DrawGLScene2);
/* Register the function called when our window is resized. */
glutReshapeFunc(&ResSizeGLScene);
/* Register the function called when the keyboard is pressed. */
glutKeyboardFunc(&keyPressed);
/* Register the function called when mouse left is clicked */
glutMouseFunc(&MouseEvent);
/* Register the function called when special keys (arrows, page down, etc) are pressed */
glutSpecialFunc(&specialKeyPressed);
/* Register the function called when mouse left is clicked */
glutMouseFunc(&MouseEvent);
/* Register the function called when special keys (arrows, page down, etc) are pressed */
glutSpecialFunc(&specialKeyPressed);
InitGL2(1280, 480);
if (DEBUG == 1){
    glutInitWindowSize(1280, 480);
    glutInitWindowPosition(0, 520);
    window3 = glutCreateWindow("LibFreeenet");
    glutDisplayFunc(&DrawGLScene);
    glutReshapeFunc(&ResSizeGLScene);
    /* Register the function called when the keyboard is pressed. */
    glutKeyboardFunc(&keyPressed);
    /* Register the function called when mouse left is clicked */
    glutMouseFunc(&MouseEvent);
    /* Register the function called when special keys (arrows, page down, etc) are pressed */
    glutSpecialFunc(&specialKeyPressed);
    glutMainLoop();
    return NULL;
}
//
void depthcb(freenect_device *dev, void *vdepth, uint32_t timestamp)
{
    int i, j;
    uint16_t *depth = (uint16_t*)vdepth;
    // save the depth map in a two dimensional array for that is easy to work with
    for (i = 0; i < 480; i++){
        for (j = 0; j < 640; j++){
            location[i][j] = depth[i*640+j];
        }
    }
    pthread_mutex_lock(&gl_backbuf_mutex);
    for (i=0; i<480; i++) {
        int pvval = location[i][i];
        int lb = pvval & 0xff;
        switch ((pvval>>8) { 
        case 0:
            depth_mid[3*i+0] = 255;
            depth_mid[3*i+1] = 255-lb;
            depth_mid[3*i+2] = 255-lb;
        break;
        case 1:
            depth_mid[3*i+0] = 255;
            depth_mid[3*i+1] = lb;
            depth_mid[3*i+2] = 0;
        break;
        case 2:
            depth_mid[3*i+0] = 255-lb;
            depth_mid[3*i+1] = 255;
            depth_mid[3*i+2] = 0;
        break;
        case 3:
            depth_mid[3*i+0] = 0;
            depth_mid[3*i+1] = 255;
            depth_mid[3*i+2] = 255-lb;
        break;
        case 4:
            depth_mid[3*i+0] = 0;
            depth_mid[3*i+1] = lb;
            depth_mid[3*i+2] = 255;
        break;
        case 5:
            depth_mid[3*i+0] = 255;
            depth_mid[3*i+1] = 255-lb;
            depth_mid[3*i+2] = 0;
        break;
        case 6:
            depth_mid[3*i+0] = lb;
            depth_mid[3*i+1] = 0;
            depth_mid[3*i+2] = 255;
        break;
        case 7:
            depth_mid[3*i+0] = 0;
            depth_mid[3*i+1] = 255;
            depth_mid[3*i+2] = lb;
        break;
        case 8:
            depth_mid[3*i+0] = 0;
            depth_mid[3*i+1] = lb;
            depth_mid[3*i+2] = 255;
        break;
        case 9:
            depth_mid[3*i+0] = 255;
            depth_mid[3*i+1] = 255-lb;
            depth_mid[3*i+2] = 0;
        break;
        case 10:
            depth_mid[3*i+0] = lb;
            depth_mid[3*i+1] = 0;
            depth_mid[3*i+2] = 255;
        break;
        case 11:
            depth_mid[3*i+0] = 0;
            depth_mid[3*i+1] = 255;
            depth_mid[3*i+2] = lb;
        break;
        case 12:
            depth_mid[3*i+0] = 255;
            depth_mid[3*i+1] = 255-lb;
            depth_mid[3*i+2] = 0;
        break;
        case 13:
            depth_mid[3*i+0] = lb;
            depth_mid[3*i+1] = 0;
            depth_mid[3*i+2] = 255;
        break;
        case 14:
            depth_mid[3*i+0] = 0;
            depth_mid[3*i+1] = 255;
            depth_mid[3*i+2] = lb;
        break;
        case 15:
            depth_mid[3*i+0] = 255;
            depth_mid[3*i+1] = 255-lb;
            depth_mid[3*i+2] = 0;
        break;
        default: break;
        }
        location[i][j] = depth_mid[i*640+j];
    }
    pthread_mutex_unlock(&gl_backbuf_mutex);
break;
        case 3:
            depth_mid[3*i+0] = 0;
            depth_mid[3*i+1] = 255;
            depth_mid[3*i+2] = 150;
            break;
        case 4:
            depth_mid[3*i+0] = 0;
            depth_mid[3*i+1] = 255-150;
            depth_mid[3*i+2] = 255;
            break;
        case 5:
            depth_mid[3*i+0] = 0;
            depth_mid[3*i+1] = 0;
            depth_mid[3*i+2] = 255-150;
            break;
        default:
            depth_mid[3*i+0] = 0;
            depth_mid[3*i+1] = 0;
            depth_mid[3*i+2] = 0;
            break;
    }
    depth += 1;
    get_depth_i+;
    pthread_cond_signal(&frame_cond);
    pthread_mutex_unlock(&backbuf_mutex);
}
void rgbcb(freeNodect_device *dev, void *rgb, uint32_t timestamp)
{
    pthread_mutex_lock(&backbuf_mutex);
    // swap buffers
    assert(rgb_back == rgb);
    rgb_back = rgb_mid;
    freeenect_set_video_buffer(dev, rgb_back);
    rgb_mid = (uint8_t*)rgb;
    freeenect_set_video_buffer(dev, rgb_back);
    pthread_mutex_unlock(&backbuf_mutex);
}
}
}

void *freeenect_threadfunc(void *arg)
{
    int accelCount = 0;
    freeenect_set_tilt_deg(f_dev, freeenect_angle);
    freeenect_set vidéomode(f_dev);
    freeenect_set_depth_callback(f_dev, depth_cb);
    freeenect_set_video_callback(f_dev, rgbcb);
    freeenect_set_video_mode(f_dev, freeenect_find_video_mode(FREENECT_RESOLUTION_MEDIUM, current_format));
    freeenect_set_depth_mode(f_dev, freeenect_find_depth_mode(FREENECT_RESOLUTION_MEDIUM, FREENECT_DEPTH_11BIT));
    freeenect_set_depth_callback(f_dev, depth_cb);
    freeenect_set_video_buffer(f_dev, rgb_back);
    freeenect_start_depth(f_dev);
    freeenect_start_video(f_dev);
    
    // print("'w'-tilt up, 's'-level, 'x'-tilt down, '0'-'6'-select LED mode, 'f'-video format\n");
    while (freeenect_process_events(&ctx) >= 0) {
        //Throttle the text output
        if (accelCount++ >= 2000)
            { accelCount = 0; freeenect_set_tilt_state(state);
            freeenect_set_depth_state(f_dev);
            state = freeenect_get_tilt_state(f_dev);
            double dx, dy, dz;
            freeenect_get_mks_accel(state, &dx, &dy, &dz);
            // printf("r raw acceleration: %4d %4d %4d mks acceleration: %4f %4f %4f", state->accelerometer_x, state->accelerometer_y, state->accelerometer_z, dx, dy, dz);
            } (flush(stdout));
    }
}
```c
if (requested_format != current_format) {
    freenect_stop_video(f_dev);
    freenect_set_video_mode(f_dev, freenect_find_video_mode(FREENECT_RESOLUTION_MEDIUM, requested_format));
    freenect_start_video(f_dev);
    current_format = requested_format;
}

// printf("\nshutting down streams...\n")
freenect_stop_depth(f_dev);
freenect_stop_video(f_dev);
freenect_release_device(f_dev);
freenect_shutdown(f_ctx);

// printf("−− done!\n")
return NULL;
}

A.3 drawX.c
This is the main draw code.
```
GLfloat LightAmbient[] = { 1.5f, 1.5f, 1.5f, 1.0f};
GLfloat LightDiffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f};
GLfloat LightPosition[] = { 1.0f, 1.0f, 2.0f, 1.0f};

void DrawGLScene2()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Clear The Screen And The Depth Buffer
    // load identity matrices
    glLoadIdentity(); // Reset The View

    // set up lights
    yrot+=xspeed; // X Axis Rotation
    xrot+=yspeed; // Y Axis Rotation
    zrot+=zspeed; // Z Axis Rotation

    if (yspeed > 0) yspeed = yspeed - 0.0001f;
    else if (yspeed < 0) yspeed = yspeed + 0.0001f;
    else if (yspeed < 0) yspeed = yspeed + 0.0001f;
    else if (yspeed < 0) yspeed = yspeed + 0.0001f;
    else if (yspeed < 0) yspeed = yspeed + 0.0001f;
    else if (yspeed < 0) yspeed = yspeed + 0.0001f;
    else if (yspeed < 0) yspeed = yspeed + 0.0001f;
    else if (yspeed < 0) yspeed = yspeed + 0.0001f;
    else if (yspeed < 0) yspeed = yspeed + 0.0001f;

    xmoon = xmoon + 0.0001f;
    ymoon = ymoon + 0.0001f;
    zmoon = zmoon + 0.0001f;

    // switch to select what texture has to be shown on the solid sphere
    switch (tyle) {
        case KEEPSPINNING: // animate
            glBindTexture(GL_TEXTURE2D, texture[2]);
            break;
        case AUTOSWITCH: // switching textures
            if (teller2 == 0) teller2 = 700;
            if (teller2 % 50 == 0)
                teller2 --;
            last++;
            if (last > 10) last = 0;
            break;
        case MANUALSWITCH: // manually switch textures
            switch (planet) {
                case 0: // mercury
                    glBindTexture(GL_TEXTURE2D, texture[0]);
                    break;
                case 1: // venus
                    glBindTexture(GL_TEXTURE2D, texture[1]);
                    break;
                case 2: // earth
                    break;
                case 3: // mars
                    break;
                case 4: // jupiter
                    break;
            }
            break;
    }
}

// white ambient light at half intensity (rgb) */
GLfloat LightAmbient[] = { 1.5f, 1.5f, 1.5f, 1.0f};
// super bright; full intensity diffuse light. */
GLfloat LightDiffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f};
// position of light [x, y, z; (position of light) ] */
GLfloat LightPosition[] = { 1.0f, 1.0f, 2.0f, 1.0f};
last = 4;
break;
case 5: // saturn
    glBindTexture(GL_TEXTURE_2D, texture[5]);
    last = 5;
    break;
case 6: // uranus
    glBindTexture(GL_TEXTURE_2D, texture[6]);
    last = 6;
    break;
case 7: // neptune
    glBindTexture(GL_TEXTURE_2D, texture[7]);
    last = 7;
    break;
case 8: // pluto
    glBindTexture(GL_TEXTURE_2D, texture[8]);
    last = 8;
    break;
case 9: // sun
    glBindTexture(GL_TEXTURE_2D, texture[9]);
    last = 9;
    break;
case 10 : // moon
    glBindTexture(GL_TEXTURE_2D, texture[46]);
    last = 46;
    break;
default :
    glBindTexture(GL_TEXTURE_2D, texture[last]);
    break;
}
break;
default :
    glBindTexture(GL_TEXTURE_2D, texture[1]); // choose the texture to use.
break;
}
// make sure globe fills the maximum amount of the screen.
// glTranslatef(-1.8f, 0.0f, z1); // move z units into the screen.
glTranslatef(0.0f, 0.0f, z1); // move z units into the screen.
glTranslatef(twoscreen*0.0f, 0.0f); // move twoscreen units to the side
GLfloat hulp = fmodf(yrot, 360.0f);

if (hulp != 0){
correcte = 1/hulp;
if (correcte > 1.0f) || correcte < -1.0f){
correcte = 0.0f;
}
if (correcte > 0.042){
correcte = 0.042;
}
if (correcte < -0.042){
correcte = -0.042;
}
}
if (DEBUG && framecounter = = 15){
printf("%f, %f, %f, %f\n", yrot, xrot, zrot, correcte);
framecounter = 0;
}
framecounter++;

if (transparent) {
    // the rotation function regarding the speed
    glRotatef(yrot, -1.0f, 0.0f, -0.42f + correcte); // Rotate On The X Axis
    glRotatef(xrot, 0.0f, 1.0f, 0.0f); // Rotate On The Y Axis
    glRotatef(zrot, -1.0f, 1.0f, -0.42f + correcte); // Rotate on the Z Axis
    glRotatef(zrotr, -1.0f, 1.0f, 0.42f + correcte); // Rotate on the Z mirror Axis

    // turn globe into 'normal position'
    glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
    glRotatef(-45.0f, 0.0f, 0.0f, 1.0f);
    glTranslatef(quadratic * 1.3f, 32.32f);
    // Draw A Sphere

    // check if overlay needs to be drawn
    if (transparent) {
        // enable alpha testing
        glEnable(GL_ALPHA_TEST);
        // enable blending
        glEnable(GL_BLEND);
// set kind of blending
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
// select texture for second bigger transparent sphere
if (animate)
{
   if (overlayproject == 42) {
      overlayproject = 20;
   }
   glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
   // choose the texture to use.
   if (interval > 10) {
      overlayproject++;
      interval = 0;
   }
   interval++;
   else {
      glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
      // choose the texture to use
   }
}

// draw a second sphere on top of the first sphere
// disable alpha testing
if (overlayproject == 20) {
   glDisable(GL_ALPHA_TEST);
   // choose the texture to use.
   if (interval > 10) {
      overlayproject++;
      interval = 0;
   }
   interval++;
   else {
      glDisable(GL_ALPHA_TEST);
   }
}

if (moon)
{
   // Rotate on the X axis
   // Rotate on the Y axis
   // Rotate on the Z axis
   glTranslatef(0.0f, 3.0f, 0.0f); // move x units into the screen.
   glRotatef(-90.0f, 1.0f, 0.0f, 0.0f); // Rotate on the X axis
   glRotatef(-moonrot, 0.0f, 1.0f, 0.0f); // Rotate on the Y axis
   glRotatef(90.0f, 1.0f, 0.0f, 0.0f); // move x units into the screen.
   glTranslatef(0.0, -3.0f, 0.0f);
   // Draw A moon
   // Rotate on the X axis
   // Rotate on the Y axis
   glDisable(GL_ALPHA_TEST);
   // Choose the texture to use
}

gluSphere(quadratic, 0.2f, 32, 32); // Draw A moon
// since this is double buffered, swap the buffers to display what just got drawn.
// set lighting
if (!light) {
   glDisable(GL_LIGHTING);
} else {
   glEnable(GL_LIGHTING);
}

glMatrixMode(GL_PROJECTION);
// set kind of bonding
glLookAt(-1.0f, 1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.707f, 0.707f, 0.0f);
// Reset The View
// Clear The Screen And The Depth Buffer
// X Axis Rotation
// Y Axis Rotation
// Z Axis Rotation
zrot+=zaspeed;
// slow down after a while.
if (zaspeed > 0) zaspeed = zaspeed - 0.0001f;
else if (zaspeed < 0) zaspeed = zaspeed + 0.0001f;
if (yspeed > 0) yspeed = yspeed + 0.0001f;
else if (yspeed < 0) yspeed = yspeed - 0.0001f;
if (zspeed > 0) zspeed = zspeed - 0.0001f;
else if (zspeed < 0) zspeed = zspeed + 0.0001f;
if (zspeedmirror > 0) zspeedmirror = zspeedmirror - 0.0001f;
else if (zspeedmirror < 0) zspeedmirror = zspeedmirror + 0.0001f;
if (xmoon > 0) xmoon = xmoon - 0.0001f;
else if (xmoon < 0) xmoon = xmoon + 0.0001f;
if (ymoon > 0) ymoon = ymoon - 0.0001f;
else if (ymoon < 0) ymoon = ymoon + 0.0001f;

bmpx2 = last;

if (yspeed > 0) yspeed = yspeed - 0.0001f;
else if (yspeed < 0) yspeed = yspeed + 0.0001f;

if (zspeed > 0) zspeed = zspeed - 0.0001f;
else if (zspeed < 0) zspeed = zspeed + 0.0001f;

if (zspeedmirror > 0) zspeedmirror = zspeedmirror - 0.0001f;
else if (zspeedmirror < 0) zspeedmirror = zspeedmirror + 0.0001f;

if (xmoon > 0) xmoon = xmoon - 0.0001f;
else if (xmoon < 0) xmoon = xmoon + 0.0001f;

if (ymoon > 0) ymoon = ymoon - 0.0001f;
else if (ymoon < 0) ymoon = ymoon + 0.0001f;

if (y + 180 < 0) y = y + 1.0f;
else if (y > 180) y = y - 1.0f;

bmpx2 = last;

// A switch to select what texture has to be shown on the solid sphere

switch (type) {
    case KEEPSPINNING:// animate
        glBindTexture(GL_TEXTURE2D, texture[2]);
        break;
    case AUTOSWITCH:// switching textures
        if (teller2 == 0) teller2 = 700;
        if (teller2 % 50 == 0) {
            teller2 =--;
            last++;}
        if (last > 10) last = 0;
        else teller2 =--;
        glBindTexture(GL_TEXTURE2D, texture[bmpx2]);
        break;
    case MANUALSWITCH:// manually switch textures
        switch (planet) {
            case 0:// mercury
                glBindTexture(GL_TEXTURE2D, texture[0]);
                last = 0;
                break;
            case 1:// venus
                glBindTexture(GL_TEXTURE2D, texture[1]);
                last = 1;
                break;
            case 2:// earth
                glBindTexture(GL_TEXTURE2D, texture[2]);
                last = 2;
                break;
            case 3:// mars
                glBindTexture(GL_TEXTURE2D, texture[3]);
                last = 3;
                break;
            case 4:// jupiter
                glBindTexture(GL_TEXTURE2D, texture[4]);
                last = 4;
                break;
            case 5:// saturn
                glBindTexture(GL_TEXTURE2D, texture[5]);
                last = 5;
                break;
            case 6:// uranus
                glBindTexture(GL_TEXTURE2D, texture[6]);
                last = 6;
                break;
            case 7:// neptune
                glBindTexture(GL_TEXTURE2D, texture[7]);
                last = 7;
                break;
            case 8:// pluto
                glBindTexture(GL_TEXTURE2D, texture[8]);
                last = 8;
                break;
            case 9:// sun
                glBindTexture(GL_TEXTURE2D, texture[9]);
                last = 9;
                break;
            case 10:// moon
                glBindTexture(GL_TEXTURE2D, texture[46]);
                last = 46;
                break;
            default:
                glBindTexture(GL_TEXTURE2D, texture[2]);
                break;
        }
break
default
bindTexture(GL_TEXTURE2D, texture[last]); // choose the texture to use.
break

// make sure globe fills the maximum amount of the screen.
translate(1.8f, 0.0f, z1); // move z units into the screen.
translate(0.0f, 0.0f, z1);  // move x units into the screen.
translate(-twoscreen, 0.0f, 0.0f); // move two screen units to the side

// the rotation function regarding the speed
rotate(yrot, -1.0f, 0.0f, 0.0f + correct);  // Rotate On The X Axis
rotate(xrot, 0.0f, 1.0f, 0.0f);  // Rotate On The Y Axis
rotate(zrot, -1.0f, 0.0f, 0.0f + correct);  // Rotate On The Z Axis
rotate(zrotmirror, 1.0f, 1.0f, -0.42f + correct); // Rotate on the Z mirror Axis

// turn globe into 'normal position'
rotate(-90.0f, 1.0f, 0.0f, 0.0f);  // Rotate On The X Axis
rotate(-ymoonrot, 0.0f, 0.0f, 0.0f);  // Rotate On The Y Axis
rotate(90.0f, 1.0f, 0.0f, 0.0f);  // Rotate On The Z Axis
translate(0.0, -3.0f, 0.0f);  // move x units into the screen.

// draw A Sphere
sphere(quadratic, 1.3f, 32, 32);

// check if overlayproject needs to be drawn
if (transp) {
  enable alpha testing
  enable blending
  enable (GL_BLEND);
  // set kind of blending
  blendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
  // select texture for second bigger transparent sphere
  if (animate) {
    if (overlayproject == 42) overlayproject = 20;
    bindTexture(GL_TEXTURE2D, texture[overlayproject]); // choose the texture to use.
    if (interval > 10) {  // overlayproject++;
      interval = 0;
    }
  }
  else {
    bindTexture(GL_TEXTURE2D, texture[overlayproject]); // choose the texture to use
    interval++;
    // draw a second sphere on top of the first sphere
    gluSphere(quadratic2, 1.4f, 32, 32);
    //disable alpha testing
    disable (GL_ALPHA.TEST);
    //disable blending
    disable (GL_BLEND);
  }
}

if (moon) {
  //glRotate(yrot, 0.0f, 0.0f, 0.0f);  // Rotate On The X Axis
  //glRotate(xrot, 0.0f, 1.0f, 0.0f);  // Rotate On The Y Axis
  //glRotate(zrot, 0.0f, 0.0f, 1.0f);  // Rotate On The Z Axis
  translate(0.0f, 3.0f, 0.0f);  // move x units into the screen.
  translate(-90.0f, 1.0f, 0.0f, 0.0f);
  //glRotate(xmoonrot, 0.0f, 1.0f, 0.0f);  // Rotate On The Y Axis
  //glRotate(ymoonrot, 0.0f, 0.0f, 0.0f);  // Rotate On The X Axis
  bindTexture(GL_TEXTURE2D, texture[47]);
  gluSphere(quadraticmoon, 0.2f, 32, 32);
  // Draw A moon
  translate(-ymoonrot, 0.0f, 1.0f, 0.0f);  // Rotate On The X Axis
  translate(ymoonrot, 0.0f, 1.0f, 0.0f);  // Rotate On The Y Axis
  translate(90.0f, 1.0f, 0.0f, 0.0f);
  translate(0.0, -3.0f, 0.0f);  // move x units into the screen.
}

// set lighting
if (!light) {
  disable (GL_LIGHTING);
} else {
  enable (GL_LIGHTING);
}
void DrawGLScene3()
{
    glutPostRedisplay();
    // since this is double buffered, swap the buffers to display what just got drawn.
    glutSwapBuffers();
}

void drawGLScene()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Clear The Screen And The Depth Buffer
    glLoadIdentity(); // Reset The View
    yrot+=xspeed; // X Axis Rotation
    xrot+=yspeed; // Y Axis Rotation
    zrot+=zspeed; // Z Axis Rotation
    xmoonrot+=xmoon; // slow down after a while.
    ymoonrot+=ymoon;
    zmoonrot+=zmoon;

    if (xspeed > 0) xspeed = xspeed - 0.0001f;
    else if (xspeed < 0) xspeed = xspeed + 0.0001f;
    if (yspeed > 0) yspeed = yspeed - 0.0001f;
    else if (yspeed < 0) yspeed = yspeed + 0.0001f;
    if (zspeed > 0) zspeed = zspeed - 0.0001f;
    else if (zspeed < 0) zspeed = zspeed + 0.0001f;
    if (xmoonrot > 0) xmoonrot = xmoonrot - 0.0001f;
    else if (xmoonrot < 0) xmoonrot = xmoonrot + 0.0001f;
    if (ymoonrot > 0) ymoonrot = ymoonrot - 0.0001f;
    else if (ymoonrot < 0) ymoonrot = ymoonrot + 0.0001f;
    if (xmoonrot > 0) ymoonrot = ymoonrot - 0.0001f;
    else if (ymoonrot < 0) ymoonrot = ymoonrot + 0.0001f;

    bmpx = last2;
    switch (type){
    case KEEPSPINNING: // animate
        glBindTexture(GL_TEXTURE2D, texture[12]);
        break;
    case AUTOSWITCH: // switching textures
        if (teller == 0) teller = 700;
        if (teller % 100 == 0){
            teller --;
            last2++;?
            if (last2 > 19) last2 = 10;
        }
        else teller --;
        glBindTexture(GL_TEXTURE2D, texture[bmpx]);
        break;
    case MANUALSWITCH: // manually change textures
        switch (planet) {
        case 0: //mercury
            glBindTexture(GL_TEXTURE2D, texture[10]);
            last2 = 10;
            break;
        case 1: //venus
            glBindTexture(GL_TEXTURE2D, texture[11]);
            last2 = 11;
            break;
        case 2: //earth
            glBindTexture(GL_TEXTURE2D, texture[12]);
            last2 = 12;
            break;
        case 3: //mars
            glBindTexture(GL_TEXTURE2D, texture[13]);
            last2 = 13;
            break;
        case 4: //jupiter
            glBindTexture(GL_TEXTURE2D, texture[14]);
            last2 = 14;
            break;
        case 5: //saturn
            glBindTexture(GL_TEXTURE2D, texture[15]);
            last2 = 15;
            break;
        case 6: //uranus
            glBindTexture(GL_TEXTURE2D, texture[16]);
            last2 = 16;
            break;
        case 7: //neptune
            glBindTexture(GL_TEXTURE2D, texture[17]);
            last2 = 17;
            break;
    }
}
break;
  case 8:  //p l u t o
    glBindTexture(GL_TEXTURE2D, texture[18]);
    last2 = 18;
    break;
  case 9:  //s u n
    glBindTexture(GL_TEXTURE2D, texture[19]);
    last2 = 19;
    break;
  default:
    glBindTexture(GL_TEXTURE2D, texture[12]);
    break;
  }
  break;

  // manually change textures

  //g l T r a n s l a t e (1.8f, 0.0f, z2) ;  // move z units into the screen, and 1.8 units to the side.
  glTranslatef(0.0f, 0.0f, z2);  // move z units into the screen, and 1.8 units to the side.
  glRotatef(180.0f, 1.0f, 0.0f, 0.0f);  // Rotate On The X Axis
  glRotatef(xrot, 0.0f, 1.0f, 0.0f);  // Rotate On The Y Axis
  glRotatef(zrot, 0.0f, 0.0f, 1.0f);  // Rotate on the Z Axis

  //f i r s t c h e c k if n e e d s t o be transparent
  if (transparent) {
    //e n a b l e a l p h a t e s t i n g
    glEnable(GL_ALPHA_TEST);
    //e n a b l e t h e b l e n d f u n c t i o n
    glEnable(GL_BLEND);
    //s e l e c t t e x t u r e f o r s e c o n d b i g g e r transparent sphere
    if (animate) {
      if (overlayproject == 42) overlayproject = 20;
      glBindTexture(GL_TEXTURE2D, texture[overlayproject]);  // choose the texture to use.
    }
    else {
      glBindTexture(GL_TEXTURE2D, texture[43]);  // choose the texture to use
    }
    //d r a w a s e c o n d s p h e r e on t o p o f t h e f i r s t s p h e r e
    gluSphere(quadratic2, 1.4f, 32, 32);
    //d i s a b l e a l p h a t e s t i n g
    glDisable(GL_ALPHA_TEST);
    //d i s a b l e b l e n d i n g
    glDisable(GL_BLEND);
  }
  else {
    glBindTexture(GL_TEXTURE2D, texture[last2]);  // choose the texture to use.
  }
}

  // set lighting
  if (light) {
    //f i r s t c h e c k if n e e d s t o be transparent
    if (transparent) {
      //e n a b l e a l p h a t e s t i n g
      glEnable(GL_ALPHA_TEST);
      //e n a b l e t h e b l e n d f u n c t i o n
      glEnable(GL_BLEND);
      //s e l e c t t e x t u r e f o r s e c o n d b i g g e r transparent sphere
      if (animate) {
        if (overlayproject == 42) overlayproject = 20;
        glBindTexture(GL_TEXTURE2D, texture[overlayproject]);  // choose the texture to use.
      }
      else {
        glBindTexture(GL_TEXTURE2D, texture[43]);  // choose the texture to use
      }
      //d r a w a s e c o n d s p h e r e on t o p o f t h e f i r s t s p h e r e
      gluSphere(quadratic2, 1.4f, 32, 32);
      //d i s a b l e a l p h a t e s t i n g
      glDisable(GL_ALPHA_TEST);
      //d i s a b l e b l e n d i n g
      glDisable(GL_BLEND);
    }
    else {
      glBindTexture(GL_TEXTURE2D, texture[43]);  // choose the texture to use
    }
    //d r a w a s e c o n d s p h e r e on t o p o f t h e f i r s t s p h e r e
    gluSphere(quadratic2, 1.4f, 32, 32);
    //d i s a b l e a l p h a t e s t i n g
    glDisable(GL_ALPHA_TEST);
    //d i s a b l e b l e n d i n g
    glDisable(GL_BLEND);
  }

  //f i r s t c h e c k if n e e d s t o be transparent
  if (moon) {
    //f i r s t c h e c k if n e e d s t o be transparent
    if (transparent) {
      //e n a b l e a l p h a t e s t i n g
      glEnable(GL_ALPHA_TEST);
      //e n a b l e t h e b l e n d f u n c t i o n
      glEnable(GL_BLEND);
      //s e l e c t t e x t u r e f o r s e c o n d b i g g e r transparent sphere
      if (animate) {
        if (overlayproject == 42) overlayproject = 20;
        glBindTexture(GL_TEXTURE2D, texture[overlayproject]);  // choose the texture to use.
      }
      else {
        glBindTexture(GL_TEXTURE2D, texture[43]);  // choose the texture to use
      }
      //d r a w a s e c o n d s p h e r e on t o p o f t h e f i r s t s p h e r e
      gluSphere(quadratic2, 1.4f, 32, 32);
      //d i s a b l e a l p h a t e s t i n g
      glDisable(GL_ALPHA_TEST);
      //d i s a b l e b l e n d i n g
      glDisable(GL_BLEND);
    }
    else {
      glBindTexture(GL_TEXTURE2D, texture[43]);  // choose the texture to use
    }
    //d r a w a s e c o n d s p h e r e on t o p o f t h e f i r s t s p h e r e
    gluSphere(quadratic2, 1.4f, 32, 32);
    //d i s a b l e a l p h a t e s t i n g
    glDisable(GL_ALPHA_TEST);
    //d i s a b l e b l e n d i n g
    glDisable(GL_BLEND);
  }

  //m a n u a l l y c h a n g e t e x t u r e s
void DrawGLScene()
{
    pthread_mutex_lock(&gl_backbuf_mutex);
    if (current_format == FREENECT_VIDEO_YUYV | 
        current_format == FREENECT_VIDEO_YUV_RGB)
    {
        glTexImage2D(GL_TEXTURE_2D, 0, 3, 640, 480, 0, GL_RGB, GL_UNSIGNED_BYTE, depth_front);
    }
    else
    {
        glTexImage2D(GL_TEXTURE_2D, 0, 1, 640, 480, 0, GL_LUMINANCE, GL_UNSIGNED_BYTE, 
                     rgb_front + 640 * 4);
    }
    pthread_mutex_unlock(&gl_backbuf_mutex);
}

// getMovement (nrofblobs, biggest);
if (current_format == FREENECT_VIDEO_RGB | 
    current_format == FREENECT_VIDEO_YUYV_RGB)
{
    glTexImage2D(GL_TEXTURE_2D, 0, 3, 640, 480, 0, GL_RGB, GL_UNSIGNED_BYTE, 
                 rgb_front);
}
else
{
    glTexImage2D(GL_TEXTURE_2D, 0, 1, 640, 480, 0, GL_LUMINANCE, GL_UNSIGNED_BYTE, 
                 rgb_front + 640 * 4);
}

// only get blobs every 2 updates
if (update == 0)
{
    kinect_getPosition();
    // update = 1;
    // }
    // update --;
}

uint8_t *tmp;

if (got_depth) {
    tmp = depth_front;
    depth_front = depth_mid;
    depth_mid = tmp;
    got_depth = 0;
}
else
    got_depth = 0;

if (got_rgb) {
    tmp = rgb_front;
    rgb_front = rgb_mid;
    rgb_mid = tmp;
    got_rgb = 0;
}
else
    got_rgb = 0;

pthread_mutex_unlock(&gl_backbuf_mutex);

// Since this is double buffered, swap the buffers to display what just got drawn.
// Swap Buffers();

// When using YUV_RGB mode, RGB frames only arrive at 15Hz, so we shouldn't force them
to draw in lock-step.
// However, this is CPU/GPU intensive when we are receiving frames in lockstep.
while (!got_depth & !got_rgb) {
    pthread_cond_wait(&frame_cond, &gl_backbuf_mutex);
}
else
    while (!got_depth || !got_rgb & requested_format != current_format) {
        pthread_cond_wait(&frame_cond, &gl_backbuf_mutex);
    }

if (requested_format != current_format) {
    pthread_mutex_unlock(&gl_backbuf_mutex);
    return;
}

// only get blobs every 2 updates
if (update == 0)
{
    kinect_getPosition();
    // update = 1;
    // }
    // update --;
}

void DrawGLScene()
{
    pthread_mutex_lock(&gl_backbuf_mutex);
    if (current_format == FREENECT_VIDEO_YUYV) {
        while (!got_depth & !got_rgb) {
            pthread_cond_wait(&frame_cond, &gl_backbuf_mutex);
        }
    }
    else
    {  
        while (!got_depth || !got_rgb & requested_format != current_format) {
            pthread_cond_wait(&frame_cond, &gl_backbuf_mutex);
        }
    }
    got_depth = 0;
    got_rgb = 0;
    pthread_mutex_unlock(&gl_backbuf_mutex);
    return;
}

// When using YUV_RGB mode, RGB frames only arrive at 15Hz, so we shouldn't force them
to draw in lock-step.
// However, this is CPU/GPU intensive when we are receiving frames in lockstep.
while (!got_depth & !got_rgb) {
    pthread_cond_wait(&frame_cond, &gl_backbuf_mutex);
}
else
    while (!got_depth || !got_rgb & requested_format != current_format) {
        pthread_cond_wait(&frame_cond, &gl_backbuf_mutex);
    }

if (requested_format != current_format) {
    pthread_mutex_unlock(&gl_backbuf_mutex);
    return;
}

// only get blobs every 2 updates
if (update == 0)
{
    kinect_getPosition();
    // update = 1;
    // }
    // update --;
}
655  glBegin(GL_TRIANGLE_FAN);
656  glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
657  glTexCoord2f(0, 0); glVertex3f(640.0, 0.0);
658  glTexCoord2f(1, 0); glVertex3f(1280.0, 0.0);
659  glTexCoord2f(1, 1); glVertex3f(1280.480, 0.0);
660  glTexCoord2f(0, 1); glVertex3f(640.480, 0.0);
661  glEnd();
662  glutSwapBuffers();
663  }
664  
665  void ReSizeGLScene2(GLsizei Width, GLsizei Height)
666  {
667  if (Height==0) { // Prevent A Divide By Zero If The Window Is Too Small
668   Height=1;
669  }
670  
671  glViewport(0, 0, Width, Height); // Reset The Current Viewport And Perspective Transformation
672  glMatrixMode(GL_PROJECTION);
673  glLoadIdentity();
674  gluPerspective(45.0f, (GLfloat)Width/(GLfloat)Height, 0.1f, 100.0f);
675  glMatrixMode(GL_MODELVIEW);
676  }
677  
678  void ReSizeGLScene(int Width, int Height){
679  if (Height==0){ // Prevent A Divide By Zero If The Window Is Too Small
680   Height=1;
681  }
682  
683  gViewport(0.0, Width, Height);
684  glMatrixMode(GL_PROJECTION);
685  glLoadIdentity();
686  glOrtho(0, 1280, 480, 0, -1.0f, 1.0f);
687  glMatrixMode(GL_MODELVIEW);
688  }
689  
690  void InitGL(int Width, int Height)
691  {
692  glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
693  glClearDepth(1.0);
694  glDepthFunc(GL_LESS);
695  glDepthMask(GL_FALSE);
696  glDisable(GL_DEPTH_TEST);
697  glDisable(GL_BLEND);
698  glDisable(GL_ALPHA_TEST);
699  glEnable(GL_TEXTURE_2D);
700  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
701  glShadeModel(GL_FLAT);
702  
703  for(int i = 1; i < MAX_TEXTURES; i++)
704  glGenTextures(1, &glTextures[i]);
705  
706  for(int i = 1; i < MAX_TEXTURES; i++)
707  glBindTexture(GL_TEXTURE_2D, glTextures[i]);
708  }
709  
710  void InitGL2(GLsizei Width, GLsizei Height) // We call this right after our OpenGL window is created.
711  {
712  LoadGLTextures(); // Load the textures
713  glEnable(GL_TEXTURE_2D); // Enable texture mapping
714  glClearColor(0.0f, 0.0f, 0.0f, 0.0f); // This Will Clear The Background Color To Black
715  glClearDepth(1.0); // Enables Clearing Of The Depth Buffer
716  glEnable(GL_DEPTH_TEST); // The Type Of Depth Test To Do
717  }
glShadeModel (GL_SMOOTH);  // Enables Smooth Color Shading

 gluPerspective (45.0f, (GLfloat)Width/(GLfloat)Height, 0.1f, 100.0f); // Calculate The Aspect Ratio Of The Window

glMatrixMode (GL_PROJECTION);
loadIdentity (); // Reset The Projection Matrix

glMatrixMode (GL_MODELVIEW);

// set up light number 1.

glLightfv (GL_LIGHT1, GL_AMBIENT, LightAmbient ); // add lighting. (ambient)

gL i g h t f v ( GL_L I G H T 1, GL_DIFFUSE, L i g h t D i f f u s e ) ; // add lighting. (diffuse).

gL i g h t f v ( GL_L I G H T 1, GL_POSITION, L i g h t P o s i t i o n ) ; // set light position.

gL i g h t e n a b l e ( GL_LIGHT1); // turn light 1 on.

/* setup blending */

gL i g h t 4 f ( 1 . 0 f , 1 . 0 f , 1 . 0 f , 0 . 2 5 ) ;
quad r a t i c = gl u N e w Q u a d r i c(); // Create A Pointer To The Quadric Object ( NEW )

quad r a t i c 2 = gl u N e w Q u a d r i c(); // Create A Pointer To The Quadric Object ( NEW )

quad r a t i c m o o n = gl u N e w Q u a d r i c(); // moon

// Can also use GLU_NONE, GLU_FLAT

quad r a t i c N or m a l s ( q u a d r a t i c, GLU_SMOOTH ); // Create Smooth Normals

quad r a t i c N or m a l s ( q u a d r a t i c 2, GLU_SMOOTH); // Create Smooth Normals

quad r a t i c N or m a l s ( quadraticmoon, GLU_SMOOTH ); // Create Smooth Normals

quad r a t i c T e x t u r e ( quadratic, GL_TRUE); // Create Texture Coords ( NEW )

quad r a t i c T e x t u r e ( quadratic2, GL_TRUE); // Create Texture Coords ( NEW )

quad r a t i c T e x t u r e ( quadraticmoon, GL_TRUE); // Create Texture Coords ( NEW )

A.4 imageload.c

load the textures into the program.

#include <stdio.h>

#include "imageload.h"

#include "libfreneckt.h"

Function Fill Array Image Names, fills an array with all the images that are used

void FillArrayImageNames () {

    filenamearray [0] = "/bin/Data/texture/mercury.bmp";
    filenamearray [1] = "/bin/Data/texture/venus.bmp";
    filenamearray [2] = "/bin/Data/texture/earth.bmp";
    filenamearray [3] = "/bin/Data/texture/mars.bmp";
    filenamearray [4] = "/bin/Data/texture/jupiter.bmp";
    filenamearray [5] = "/bin/Data/texture/saturn.bmp";
    filenamearray [6] = "/bin/Data/texture/uranus.bmp";
    filenamearray [7] = "/bin/Data/texture/neptune.bmp";
    filenamearray [8] = "/bin/Data/texture/pluto.bmp";
    filenamearray [9] = "/bin/Data/texture/sun.bmp";
    filenamearray [10] = "/bin/Data/texture/mercury2.bmp";
    filenamearray [12] = "/bin/Data/texture/earth2.bmp";
    filenamearray [13] = "/bin/Data/texture/mars2.bmp";
    filenamearray [14] = "/bin/Data/texture/jupiter2.bmp";
    filenamearray [15] = "/bin/Data/texture/saturn2.bmp";
    filenamearray [16] = "/bin/Data/texture/uranus2.bmp";
    filenamearray [17] = "/bin/Data/texture/neptune2.bmp";
    filenamearray [18] = "/bin/Data/texture/pluto2.bmp";
    filenamearray [19] = "/bin/Data/texture/sun2.bmp";
    filenamearray [20] = "/bin/Data/texture/overlay/1.bmp";
    filenamearray [21] = "/bin/Data/texture/overlay/2.bmp";
    filenamearray [22] = "/bin/Data/texture/overlay/3.bmp";
    filenamearray [23] = "/bin/Data/texture/overlay/4.bmp";
    filenamearray [24] = "/bin/Data/texture/overlay/5.bmp";
    filenamearray [25] = "/bin/Data/texture/overlay/6.bmp";
    filenamearray [26] = "/bin/Data/texture/overlay/7.bmp";
    filenamearray [27] = "/bin/Data/texture/overlay/8.bmp";
    filenamearray [28] = "/bin/Data/texture/overlay/9.bmp";
    filenamearray [29] = "/bin/Data/texture/overlay/10.bmp";
    filenamearray [30] = "/bin/Data/texture/overlay/11.bmp";
    filenamearray [31] = "/bin/Data/texture/overlay/12.bmp";
    filenamearray [32] = "/bin/Data/texture/overlay/13.bmp";
    filenamearray [33] = "/bin/Data/texture/overlay/14.bmp";
    filenamearray [34] = "/bin/Data/texture/overlay/15.bmp";
    filenamearray [35] = "/bin/Data/texture/overlay/16.bmp";
    filenamearray [36] = "/bin/Data/texture/overlay/17.bmp";
    filenamearray [37] = "/bin/Data/texture/overlay/18.bmp";
    filenamearray [38] = "/bin/Data/texture/overlay/19.bmp";
    filenamearray [39] = "/bin/Data/texture/overlay/20.bmp";
    filenamearray [40] = "/bin/Data/texture/overlay/21.bmp";
    filenamearray [41] = "/bin/Data/texture/overlay/22.bmp";
    filenamearray [42] = "/bin/Data/texture/overlay/23.bmp";
    filenamearray [43] = "/bin/Data/texture/overlay/24.bmp";
    filenamearray [44] = "/bin/Data/texture/overlay/25.bmp";
    filenamearray [45] = "/bin/Data/texture/overlay/26.bmp";
    filenamearray [46] = "/bin/Data/texture/overlay/27.bmp";
    filenamearray [47] = "/bin/Data/texture/overlay/28.bmp";
    filenamearray [48] = "/bin/Data/texture/overlay/29.bmp";
    filenamearray [49] = "/bin/Data/texture/overlay/30.bmp";
    filenamearray [50] = "/bin/Data/texture/overlay/31.bmp";
    filenamearray [51] = "/bin/Data/texture/overlay/32.bmp";
    filenamearray [52] = "/bin/Data/texture/overlay/33.bmp";
    filenamearray [53] = "/bin/Data/texture/overlay/34.bmp";
    filenamearray [54] = "/bin/Data/texture/overlay/35.bmp";
    filenamearray [55] = "/bin/Data/texture/overlay/36.bmp";
    filenamearray [56] = "/bin/Data/texture/overlay/37.bmp";
    filenamearray [57] = "/bin/Data/texture/overlay/38.bmp";
    filenamearray [58] = "/bin/Data/texture/overlay/39.bmp";
    filenamearray [59] = "/bin/Data/texture/overlay/40.bmp";
}
```c
filenamarray[23] = "./bin/data/texture/overlay/1.bmp";
filenamarray[24] = "./bin/data/texture/overlay/2.bmp";
filenamarray[25] = "./bin/data/texture/overlay/3.bmp";
filenamarray[26] = "./bin/data/texture/overlay/4.bmp";
filenamarray[27] = "./bin/data/texture/overlay/5.bmp";
filenamarray[28] = "./bin/data/texture/overlay/6.bmp";
filenamarray[29] = "./bin/data/texture/overlay/7.bmp";
filenamarray[31] = "./bin/data/texture/overlay/9.bmp";
filenamarray[32] = "./bin/data/texture/overlay/10.bmp";
filenamarray[33] = "./bin/data/texture/overlay/11.bmp";
filenamarray[34] = "./bin/data/texture/overlay/12.bmp";
filenamarray[35] = "./bin/data/texture/overlay/13.bmp";
filenamarray[36] = "./bin/data/texture/overlay/14.bmp";
filenamarray[37] = "./bin/data/texture/overlay/15.bmp";
filenamarray[38] = "./bin/data/texture/overlay/16.bmp";
filenamarray[39] = "./bin/data/texture/overlay/17.bmp";
filenamarray[40] = "./bin/data/texture/overlay/18.bmp";
filenamarray[41] = "./bin/data/texture/overlay/19.bmp";
filenamarray[42] = "./bin/data/texture/overlay/20.bmp";
filenamarray[43] = "./bin/data/texture/overlay/21.bmp";
filenamarray[44] = "./bin/data/texture/overlay/22.bmp";
filenamarray[45] = "./bin/data/texture/overlay/23.bmp";
filenamarray[46] = "./bin/data/texture/overlay/test1.bmp";
filenamarray[47] = "./bin/data/texture/overlay/test2.bmp";
filenamarray[48] = "./bin/data/texture/overlay/moon.bmp";
filenamarray[49] = "./bin/data/texture/overlay/moon2.bmp";
filenamarray[50] = "./bin/data/texture/overlay/old/new.bmp";
filenamarray[51] = "./bin/data/texture/overlay/new.bmp";

int x = 0;

while (x < MAXIMAGES)
{
    FILE *file;
    image[x] = (Image *)malloc(sizeof(Image));
    if (image[x] == NULL) {
        printf("Error allocating space for image\n", x);
        exit(0);
    }
    if (ImageLoad(filenamarray[x], image[x]) == 0) {
        exit(1);
    }
    // create Texture
    glGenTextures(3, &texture[x]);
    // texture 1 (linear scaling)
    glBindTexture(GL_TEXTURE2D, texture[x]); // 2D texture (x and y size)
    glTexParameteri(GL_TEXTURE2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    // scale linearly when image bigger than texture
    glTexParameteri(GL_TEXTURE2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    // scale linearly when image smaller than texture
    glTexImage2D(GL_TEXTURE2D, 0, 3, image[x]->sizeX, image[x]->sizeY, 0, GL_RGB,
              GL_UNSIGNED_BYTE, image[x]->data);
    // 2D texture, 3 colors, width, height, RGB in that order, byte data, and the data.
    glBuild2DMipmaps(GL_TEXTURE2D, 3, image[x]->sizeX, image[x]->sizeY, GL_RGB,
                     GL_UNSIGNED_BYTE, image[x]->data);
    x++;
}
```

// Load a texture from .BMP file
// reads image header to see if image is valid
// has to be 24 bit bmp

int ImageLoad(char *filename, Image *image)
{
    FILE *file;
    unsigned long size; // size of the image in bytes.
    unsigned long i; // standard counter.
    unsigned short int planes; // number of planes in image (must be 1)
    unsigned short int bpp; // number of bits per pixel (must be 24)
    char temp; // used to convert bgr to rgb color.
    // Make sure the file exists

    // Make sure the file exists
```
A.5 interact.c

load the textures into the program
```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>  // Header for sleeping

int freenect_angle = 10;
int freenectLed;

void keyPressed(unsigned char key, int x, int y)
{
    /* to avoid thrashing this procedure */
    usleep(100);
    // printf("key = %d\n", key);
    switch (key)
    {
    case 'Q':
    case 'q':
    case 'ESCAPE':  // kill everything.
    case ' ':
    case 'Q':
        if (program == GLOBES)
        {
            printf("escaping program=Holokin\n");
            // kill the windows
            glutDestroyWindow(window3);
            glutDestroyWindow(window);
            glutDestroyWindow(window2);
        }
    break;

    case 'l':
    case 'L':
        // switch the lighting.
    light = light ? 0 : 1;
    break;

    case 't':
    case 'T':
        transparent = !transparent;
    break;

    case 'c':
    case 'C':
        break;
```
if (type != MANUALSWITCH) type = MANUALSWITCH;
else type = DEFAULTVIEW;
break;
case '+' :
  planet++; 
  if (planet == 11) planet = 0;
  break;
case '-' :
  planet--; 
  if (planet == -1) planet = 10;
  break;
case KEYPAD_UP: //numpad up 8
  xspeed -= 0.01f;
  break;
case KEYPAD_DOWN: //numpad down 2
  xspeed += 0.01f;
  break;
case KEYPAD_LEFT: //numpad left 4
  yspeed = 0.0f;
  break;
case KEYPAD_RIGHT: //numpad right 6
  yspeed += 0.01f;
  break;
case KEYPAD_MIDDLE : // numpad centre 5
  yspeed = 0.0f;
  xspeed = 0.0f;
  zspeed = 0.0f;
  zspeedmirror = 0.0f;
  type = DEFAULTVIEW;
  break;
case KEYPAD_LEFT_UP: // numpad 7
  zspeed -= 0.01f;
  break;
case KEYPAD_RIGHT_UP : // numpad 9
  zspeed += 0.01f;
  break;
case KEYPAD_LEFT_DOWN: // numpad 9
  zspeedmirror -= 0.01f;
  break;
case KEYPAD_RIGHT_DOWN : // numpad 9
  zspeedmirror += 0.01f;
  break;
case 'w' :
  freenect_angle ++;
  if (freenect_angle > 30) { 
    freenect_angle = 30;
}
  freenect_set_tilt_legs(fdev, freenect_angle);
  break;
case 'a' :
  freenect_angle = 0;
  freenect_set_tilt_legs(fdev, freenect_angle);
  break;
case 's' :
  freenect_angle --;
  if (freenect_angle < -30) { 
    freenect_set_tilt_legs(fdev, freenect_angle);
}
  break;
case 'd' :
  freenect_set_tilt_legs(fdev, freenect_angle);
  break;
case ']' :
  if (cycle == 0) freenect_set_led(fdev, LED_OFF);
  if (cycle == 1) freenect_set_led(fdev, LED_GREEN);
  if (cycle == 2) freenect_set_led(fdev, LED_YELLOW);
  if (cycle == 3) freenect_set_led(fdev, LED_RED);
  if (cycle == 4) freenect_set_led(fdev, LED_YELLOWGREEN);
  if (cycle == 5) freenect_set_led(fdev, LED_YELLOWRED);
  if (cycle == 6) freenect_set_led(fdev, LED_YELLOWGREENYELLOW);
  break;
155 if (cycle == 1) freeenct set led(fdev,LEDGREEN);
156 cycle++;
157 if (cycle == 7) cycle = 0;
158 case 'm':
159     //moon = !moon;
160 break;
161 default:
162     printf ("Key %d pressed. No action there yet.\n", key);
163 break;
164 }
165
166 /∗ The function called whenever a normal key is pressed. */
167 void specialKeyPressed(int key, int x, int y)
168 { /* avoid thrashing this procedure */
169    usleep(100);
170    switch (key) {
171 case GLUT_KEY_PAGEUP: // move the cube into the distance.
172        z1 -=0.02f;
173 break;
174 case GLUT_KEY_PAGEDOWN: // move the cube closer.
175        z1 +=0.02f;
176 break;
177 case GLUT_KEY_UP: // decrease x rotation speed;
178        x2 -=0.02f;
179 break;
180 case GLUT_KEY_DOWN: // increase x rotation speed;
181        x2 +=0.02f;
182 break;
183 case GLUT_KEY_LEFT: // decrease y or z rotation speed;
184        xmoon +=0.01f;
185 break;
186 case GLUT_KEY_RIGHT: // increase y or z rotation speed;
187        xmoon -=0.01f;
188 break;
189 case GLUT_KEY_END:
190    yspeed = 0.0f;
191    xspeed = 0.0f;
192    zspeed = 0.0f;
193    zspeedmirror = 0.0f;
194    yrot = 0;
195    xrot = 0;
196    zrot = 0;
197    zrotmirror = 0;
198    xmoon = 0 0f;
199    ymoon = 0.0f;
200    typemirror = DEFAULTVIEW;
201    animate = GL_FALSE;
202 break;
203    case GLUT_KEY_HOME:
204 if (type != KEEPSPINNING) {
205        type = KEEPSPINNING;
206        animate = !animate;
207    }
208 else {
209        type = DEFAULTVIEW;
210        animate = !animate;
211    }
212 break;
213 default:
214 break;
215    }
216    case GLUT_KEY_HOME:
217 if (type != KEEPSPINNING) {
218        type = KEEPSPINNING;
219        animate = !animate;
220    }
221 else {
222        type = DEFAULTVIEW;
223        animate = !animate;
224    }
225 break;
226 default:
227 break;
228    }
229 void MouseEvent(int button, int state, int x, int y){
230
231
232     key_modifiers = glutGetModifiers();
233     if (button == GLUT_LEFT_BUTTON) {
234         if (state == GLUT_DOWN) {
235             moving = 1;
236             start_x = x;
237             start_y = y;
238         }
239         if (state == GLUT_UP) {
240             moving = 0;
241         }
242         glutPostRedisplay();
243     }
244     }
245     void MouseMotion(int x, int y){
246         static double acc;
247         if (moving){
248             if (keyModifiers & GLUT_ACTIVE_CTRL) acc = 0.02; else acc = 1.0;
249             if (!animate){
250                 yrot -= (double)(y-start_y)*acc;
251                 xrot += (double)(x-start_x)*acc;
252             } else {
253                 yspeed = (double)(y-start_y)*0.05f;
254                 xspeed = (double)(x-start_x)*0.05f;
255             }
256             start_x = x;
257             start_y = y;
258         }
259         glutPostRedisplay();
260     }

A.6  kinect.c

Kinect code, to register input

#include <stdio.h>
#include <stdlib.h>
#include "libfreenect.h"
#include "kinect.h"

void kinect_set_active_area(int xlow, int xhigh, int ylow, int yhigh, int zlow, int zhigh)
{
    xlowerbound = xlow;
    xupperbound = xhigh;
    ylowerbound = ylow;
    yupperbound = yhigh;
    zlowerbound = zlow;
    zupperbound = zhigh;
}

void kinect_get_position()
{
    int i, j;
    for (i = 0; i < 480; i++){
        for (j = 0; j < 640; j++){
            // First define if in target area
            if (location[i][j] < zupperbound || location[i][j] > zlowerbound)
                loc[i][j] = 1;
        }
    }
    else{
        loc[i][j] = 0;
    }
    getBlobs();
}

void kinect_get_state()
```c
void getMovement()
{
    static double acc = 0.3;
    int reset;
    // printf(" moving: %i", moving);
    if (!moving)
        if (meanx == meanxoud && meany == meanyoud && meanx != 0 && meany != 0)
            if (!animate && teller3 > 20)
                teller3 = 0;
            moving = 1;
        else if (teller3 > 10)
            teller3 = 0;
        moving = 1;
        teller3++;
        // printf(" teller3: %i", teller3);
    else if (moving)
        if (meanx == 0 && meany == 0)
            if (teller3 > 40)
                teller3 = 0;
            moving = 1;
        else
            teller3++;
        // printf(" teller3+: %i", teller3);
    else
        if (meanxoud != 0 && meanyoud != 0)
            if (meanxoud - meanx < 0)
                xsped = -0.2f;
            reset = 0;
        else if (meanxoud - meanx > 0)
            xsped = 0.2f;
        reset = 0;
    else if (meanyoud - meany > 0)
        ysped = -0.2f;
    else if (meanyoud - meany < 0)
        ysped = +0.2f;
    reset = 0;
    // if (meanxoud == meanx && meanyoud == meany)
    // if (reset == 5)
    // ysped = -0;
    // ysped = 0;
    // ysped = 0;
    // reset = 0;
    // else reset++;
    // if ((means == 0 || meany == 0) && (meanxoud == 0 || meanxoud == 0) && !animate)
    // stop rotation
    xsped = 0;
    ysped = 0;
    moving = 0;
}
```
void emptyarray()
{
    int i, j;
    // empty arrays
    for (i = 0; i < 240; i++)
    {
        for (j = 0; j < 320; j++)
        {
            smallblocks[i][j] = 0;
        }
    }
    for (i = 0; i < 120; i++)
    {
        for (j = 0; j < 160; j++)
        {
            mediumblocks[i][j] = 0;
        }
    }
    for (i = 0; i < 60; i++)
    {
        for (j = 0; j < 80; j++)
        {
            largeblocks[i][j] = 0;
        }
    }
    for (i = 0; i < 30; i++)
    {
        for (j = 0; j < 40; j++)
        {
            bigblocks[i][j] = 0;
        }
    }
    for (i = 0; i < 15; i++)
    {
        for (j = 0; j < 20; j++)
        {
            hugeblocks[i][j] = 0;
        }
    }
    for (i = 0; i < 15; i++)
    {
        for (j = 0; j < 15; j++)
        {
            blob[i][j] = 0;
        }
    }
}

void getBlobs()
{
    printf(" frame++
\n") ;
    int i, j;
    int temp = 0;
    int count = 0;
    for (i = 0; i < 480; i++)
    {
        for (j = 0; j < 640; j++)
        {
            if (loc[i][j] != 1) count++;
        }
    }
    printf(" count=\n", count);
    count = 0;
    // first make square blocks
    for (i = 0; i < 240; i++)
    {
        for (j = 0; j < 320; j++)
        {
            if (loc[2*i][2*j] != 1) temp++;
        }
    }
    if (temp > 1)
    {
        smallblocks[i][j] = 1;
        temp = 0;
    } else
    {
        smallblocks[i][j] = 0;
        temp = 0;
    }
}
for (i = 0; i < 240; i++) {
    for (j = 0; j < 320; j++) {
        if (smallblocks[i][j] == 1) count++;
    }
    printf("smallcount=%i\n", count);
    count = 0;
}

// make the blocks bigger

temp = 0;

for (i = 0; i < 120; i++) {
    for (j = 0; j < 160; j++) {
        if (smallblocks[2*i][2*j] == 1) {
            temp++;
        }
    }
    if (temp > 1) {
        mediumblocks[i][j] = 1;
        temp = 0;
    } else {
        mediumblocks[i][j] = 0;
        temp = 0;
    }
}

for (i = 0; i < 120; i++) {
    for (j = 0; j < 160; j++) {
        if (mediumblocks[i][j] == 1) count++;
    }
    printf("mediumcount=%i\n", count);
    count = 0;
}

// make the blocks bigger

temp = 0;

for (i = 0; i < 60; i++) {
    for (j = 0; j < 80; j++) {
        if (mediumblocks[2*i][2*j] == 1) {
            temp++;
        }
    }
    if (temp > 1) {
        largeblocks[i][j] = 1;
        temp = 0;
    } else {
        largeblocks[i][j] = 0;
        temp = 0;
    }
}

for (i = 0; i < 60; i++) {
    for (j = 0; j < 80; j++) {
        if (largeblocks[i][j] == 1) count++;
    }
    printf("largecount=%i\n", count);
    count = 0;
}

// make the blocks bigger
temp = 0;
for (i = 0; i < 30; i++){
    temp++;
    if (largeblocks[2*i][2*j] == 1){
        temp++;
        if (largeblocks[2*i+1][2*j] == 1){
            temp++;
            if (largeblocks[2*i+1][2*j+1] == 1){
                temp++;
                if (temp > 2){
                    bigblocks[i][j] = 1;
                    temp = 0;
                }
            }
            else{
                bigblocks[i][j] = 0;
                temp = 0;
            }
        }
    }
}
for (i = 0; i < 30; i++){
    if (temp > 2){
        bigblocks[i][j] = 1;
        temp = 0;
    }
}
else{
    bigblocks[i][j] = 0;
    temp = 0;
}
for (i = 0; i < 30; i++){
    for (j = 0; j < 40; j++){
        if (largeblocks[2*i][2*j] == 1){
            temp++;
            if (largeblocks[2*i+1][2*j] == 1){
                temp++;
                if (largeblocks[2*i+1][2*j+1] == 1){
                    temp++;
                    if (temp > 2){
                        hugeblocks[i][j] = 1;
                        temp = 0;
                    }
                }
            }
        }
    }
}
for (i = 0; i < 30; i++){
    for (j = 0; j < 40; j++){
        if (largeblocks[2*i][2*j] == 1){
            temp++;
            if (largeblocks[2*i+1][2*j] == 1){
                temp++;
                if (largeblocks[2*i+1][2*j+1] == 1){
                    temp++;
                    if (temp > 2){
                        hugeblocks[i][j] = 1;
                        temp = 0;
                    }
                }
            }
        }
    }
}
for (i = 0; i < 30; i++){
    if (temp > 2){
        bigblocks[i][j] = 1;
        temp = 0;
    }
}
else{
    bigblocks[i][j] = 0;
    temp = 0;
}
}
for (i = 0; i < 30; i++){
    if (temp > 2){
        bigblocks[i][j] = 1;
        temp = 0;
    }
}
else{
    bigblocks[i][j] = 0;
    temp = 0;
}
}
for (i = 0; i < 30; i++){
    if (temp > 2){
        bigblocks[i][j] = 1;
        temp = 0;
    }
}
else{
    bigblocks[i][j] = 0;
    temp = 0;
}
}
for (i = 0; i < 30; i++){
    if (temp > 2){
        bigblocks[i][j] = 1;
        temp = 0;
    }
}
else{
    bigblocks[i][j] = 0;
    temp = 0;
}
}
// now we throw away the left most 2 bits and the right most 3 bits
for (i = 0; i < 15; i++){
    for (j = 0; j < 20; j++){
        if (hugeblocks[i][j] == 1) count++;
    }
    printf("hugecount=%i\n", count);
    count = 0;
}
// now we throw away the left most 2 bits and the right most 3 bits
for (i = 0; i < 15; i++){
    for (j = 0; j < 20; j++){
        if (hugeblocks[i][j] == 1) count++;
    }
    printf("hugecount=%i\n", count);
    count = 0;
}
// now we throw away the left most 2 bits and the right most 3 bits
for (i = 0; i < 15; i++){
    for (j = 0; j < 20; j++){
        if (hugeblocks[i][j] == 1) count++;
    }
    printf("hugecount=%i\n", count);
    count = 0;
}
// now we throw away the left most 2 bits and the right most 3 bits
for (i = 0; i < 15; i++){
    for (j = 0; j < 20; j++){
        if (hugeblocks[i][j] == 1) count++;
    }
    printf("hugecount=%i\n", count);
    count = 0;
}
```c
for (j = 0; j < 16; j++) {
    column[j] = 0;
}
for (i = 0; i < 15; i++) {
    row[i] = 1;
    column[j] = 1;
}

printf("row: ");
for (i = 0; i < 15; i++) {
    printf("%i, ", row[i]);
}
printf("\ncolumn: ");
for (j = 0; j < 15; j++) {
    printf("%i", column[j]);
}
printf("\n");
count = 0;
meanx = 0;
meany = 0;

// now we have values we can track
// calculate mean of x
for (i = 0; i < 15; i++) {
    if (row[i] == 1) {
        count++;
        meanx = meanx + i;
    }
}
if (count != 0) {
    meanx = meanx / count;
}
count = 0;

// calculate mean of y
for (j = 0; j < 15; j++) {
    if (column[j] == 1) {
        count++;
        meany = meany + j;
    }
}
if (count != 0) {
    meany = meany / count;
}

printf("meanx = %i meany = %i\n", meanx, meany);
printf("meanxoud = %i meanyoud = %i\n", meanxoud, meanyoud);
gMoveMent();
```

A.7 kubus2.c

code to show a cube that can be manipulated by the kinect

```c
#include <stdio.h>
#include <stdlib.h>
#include "libfreenect.h"
#include "kinect.h"

void kinect_set_active_area(int xlow, int xhigh, int ylow, int yhigh, int zlow, int zhigh)
{
    xlowerbound = xlow;
    xupperbound = xhigh;
    ylowerbound = ylow;
    yupperbound = yhigh;
```

47
void get_position()
{
    int i, j;
    for (i = 0; i < 480; i++) {
        for (j = 0; j < 640; j++)
            if ((location[i][j] < zupperbound) || (location[i][j] > zlowerbound))
                loc[i][j] = 1;
            else
                loc[i][j] = 0;
    }
    getBlobs();
}

void get_state()
{
    if (!moving)
        if (meanx == meanxoud && meany == meanyoud && meanx != 0 && meany != 0)
            if (!animate && teller3 > 20)
                telle3 = 0;
            moving = 1;
        else if (teller3 > 10)
            telle3 = 0;
        moving = 1;
    else
        telle3++;
    else if (!animate)
        xrot = (double)(meany - meanyoud) * acc + 2;
        yrot = (double)(meanx - meanxoud) * acc + 2;
    if (animate) 
        if (meanxoud != 0 && meanx != 0)
            if (meanxoud == 0 && meanx < 0)
                xspeed -= 0.2f;
            else
                if (meanxoud - meanx < 0)
                    xspeed += 0.2f;
                if (meanxoud != 0 && meanx != 0)
                    if (meanyoud != 0 && meany > 0)
                        yspeed += 0.2f;
                    else if (meanyoud != 0)
                        yspeed -= 0.2f;
                reset = 0;
            } else 
                if (meanxoud == 0 && meanx < 0)
                    xspeed -= 0.2f;
                } else 
                    if (meanxoud - meanx < 0)
                        yspeed += 0.2f; 
                    reset = 0; 
            } 
} 

48
91 } } } 
92 } if (meanxoud == meanx && meanyoud == meany) {
93 } if (reset == 5) {
94 } ifspeed = 0; 
95 } ifspeed = 0; 
96 } reset = 0; 
97 } else reset++; 
98 } 
99 } 
100 } if ((meanx = 0 || meany = 0) && (meanyoud = 0 || meanxoud = 0) && !animate) {
101 } stop rotation 
102 } exspeed = 0; 
103 } yspeed = 0; 
104 } moving = 0; 
105 } 
106 } meanxoud = meanx; 
107 } meanyoud = meany; 
108 } 
109 } emptyarray() { 
110 } int i, j; 
111 } empty arrays 
112 } for (i = 0; i < 240; i++) {
113 } for (j = 0; j < 320; j++) {
114 } smallblocks[i][j] = 0; 
115 } 
116 } for (i = 0; i < 120; i++) {
117 } for (j = 0; j < 160; j++) {
118 } mediumblocks[i][j] = 0; 
119 } 
120 } for (i = 0; i < 60; i++) {
121 } for (j = 0; j < 80; j++) {
122 } largeblocks[i][j] = 0; 
123 } 
124 } for (i = 0; i < 30; i++) {
125 } for (j = 0; j < 40; j++) {
126 } bigblocks[i][j] = 0; 
127 } 
128 } for (i = 0; i < 15; i++) {
129 } for (j = 0; j < 15; j++) {
130 } blob[i][j] = 0; 
131 } 
132 } for (i = 0; i < 480; i++) {
133 } for (j = 0; j < 640; j++) {
134 } loc[i][j] != 1) count++; 
135 } 
136 } printf("frame++\n"); 
137 } printf("count=\n", count); 
138 } count = 0; 
139 } first make square blocks 
140 } for (i = 0; i < 240; i++) {
141 } for (j = 0; j < 320; j++) {
if (loc[2*i][2*j] != 1) {
    temp++;
}
if (loc[2*i+1][2*j] != 1) {
    temp++;
}
if (loc[2*i][2*j+1] != 1) {
    temp++;
}
if (loc[2*i+1][2*j+1] != 1) {
    temp++;
}
if (temp > 1) {
    smallblocks[i][j] = 1;
    temp = 0;
}
else {
    smallblocks[i][j] = 0;
    temp = 0;
}
for (i = 0; i < 240; i++) {
    for (j = 0; j < 320; j++) {
        if (smallblocks[i][j] == 1) count++;  
    }
    printf("smallcount=\n", count);
    count = 0;
}
for (i = 0; i < 120; i++) {
    for (j = 0; j < 160; j++) {
        if (smallblocks[2*i][2*j] == 1) 
            temp++;
        if (smallblocks[2*i+1][2*j] == 1) 
            temp++;
        if (smallblocks[2*i][2*j+1] == 1) 
            temp++;
        if (smallblocks[2*i+1][2*j+1] == 1) 
            temp++;
    }
    if (temp > 1) {
        mediumblocks[i][j] = 1;
        temp = 0;
    }
    else {
        mediumblocks[i][j] = 0;
        temp = 0;
    }
}
for (i = 0; i < 120; i++) {
    for (j = 0; j < 160; j++) {
        if (mediumblocks[2*i][2*j] == 1) 
            temp++;
        if (mediumblocks[2*i+1][2*j] == 1) 
            temp++;
        if (mediumblocks[2*i][2*j+1] == 1) 
            temp++;
        if (mediumblocks[2*i+1][2*j+1] == 1) 
            temp++;
    }
    printf("mediumcount=\n", count);
    count = 0;
}
for (i = 0; i < 60; i++) {
    for (j = 0; j < 80; j++) {
        if (mediumblocks[2*i][2*j] == 1) 
            temp++;
        if (mediumblocks[2*i+1][2*j] == 1) 
            temp++;
        if (mediumblocks[2*i][2*j+1] == 1) 
            temp++;
        if (mediumblocks[2*i+1][2*j+1] == 1) 
            temp++;
    }
    printf("mediumcount=\n", count);
    count = 0;
}
// make the blocks bigger
for (i = 0; i < 120; i++) {
    for (j = 0; j < 160; j++) {
        if (smallblocks[2*i][2*j] == 1) 
            temp++;
        if (smallblocks[2*i+1][2*j] == 1) 
            temp++;
        if (smallblocks[2*i][2*j+1] == 1) 
            temp++;
        if (smallblocks[2*i+1][2*j+1] == 1) 
            temp++;
    }
    if (temp > 1) {
        mediumblocks[i][j] = 1;
        temp = 0;
    }
    else {
        mediumblocks[i][j] = 0;
        temp = 0;
    }
}
for (i = 0; i < 120; i++) {
    for (j = 0; j < 160; j++) {
        if (mediumblocks[2*i][2*j] == 1) 
            temp++;
        if (mediumblocks[2*i+1][2*j] == 1) 
            temp++;
        if (mediumblocks[2*i][2*j+1] == 1) 
            temp++;
        if (mediumblocks[2*i+1][2*j+1] == 1) 
            temp++;
    }
    printf("mediumcount=\n", count);
    count = 0;
}
// make the blocks bigger
for (i = 0; i < 60; i++) {
    for (j = 0; j < 80; j++) {
        if (mediumblocks[2*i][2*j] == 1) 
            temp++;
        if (mediumblocks[2*i+1][2*j] == 1) 
            temp++;
        if (mediumblocks[2*i][2*j+1] == 1) 
            temp++;
        if (mediumblocks[2*i+1][2*j+1] == 1) 
            temp++;
    }
    printf("mediumcount=\n", count);
    count = 0;
}
```c
if (mediumblocks[2*i + 1][2*j + 1] == 1)
    temp++;
if (temp > 1)
    largeblocks[i][j] = 1;
temp = 0;
else
    largeblocks[i][j] = 0;
temp = 0;
}
for (i = 0; i < 60; i++)
    for (j = 0; j < 80; j++)
        if (largeblocks[i][j] == 1) count++;
printf("large count = %i \n", count);
count = 0;

// make the blocks bigger
for (i = 0; i < 30; i++)
    for (j = 0; j < 40; j++)
        if (largeblocks[2*i][2*j] == 1)
            temp++;
        if (largeblocks[2*i + 1][2*j] == 1)
            temp++;
        if (largeblocks[2*i][2*j + 1] == 1)
            temp++;
        if (largeblocks[2*i + 1][2*j + 1] == 1)
            temp++;
if (temp > 2)
    bigblocks[i][j] = 1;
temp = 0;
else
    bigblocks[i][j] = 0;
temp = 0;
}
for (i = 0; i < 30; i++)
    for (j = 0; j < 20; j++)
        if (bigblocks[i][j] == 1) count++;
printf("big count = %i \n", count);
count = 0;

// big blocks
for (i = 0; i < 15; i++)
    for (j = 0; j < 20; j++)
        if (bigblocks[2*i][2*j] == 1)
            temp++;
        if (bigblocks[2*i + 1][2*j] == 1)
            temp++;
        if (bigblocks[2*i][2*j + 1] == 1)
            temp++;
        if (bigblocks[2*i + 1][2*j + 1] == 1)
            temp++;
if (temp > 1)
    hugeblocks[i][j] = 1;
temp = 0;
else
    hugeblocks[i][j] = 0;
```
```c
322 temp = 0;
323 }
324 }
325 }
326 for (i = 0; i < 15; i++){
327 for (j = 0; j < 20; j++){
328 if (hugeblocks[i][j] == 1) count++;
329 }
330 }
331 printf("hugecount=\n", count);
332 count = 0;
333 //now we throw away the left most 2 bits and the rightmost 3 bits
334 for(i = 0; i < 15; i++){
335 for (j = 0; j < 15; j++){
336 blob[i][j] = hugeblocks[i+2][j];
337 }
338 }
339 }
340 int row[15];
341 int column[16];
342 for (i = 0; i < 15; i++){
343 row[i] = 0;
344 }
345 for (j = 0; j < 16; j++){
346 column[j] = 0;
347 }
348 for (i = 0; i < 15; i++){
349 for (j = 0; j < 16; j++){
350 if (hugeblocks[i][j] == 1){
351 row[i] = 1;
352 column[j] = 1;
353 }
354 }
355 }
356 printf("row:");
357 for (i = 0; i < 15; i++){
358 printf("%i, ", row[i]);
359 }
360 printf(\n");
361 printf("column:");
362 for (j = 0; j < 16; j++){
363 printf("%i, ", column[j]);
364 }
365 printf(\n");
366 count = 0;
367 meanx = 0;
368 meany = 0;
369 //now we have values we can track
370 //calculate mean of x
371 for (i = 0; i < 15; i++){
372 if (row[i] == 1){
373 count++;
374 meanx = meanx + i;
375 }
376 }
377 if (count != 0){
378 meanx = meanx/count;
379 }
380 count = 0;
381 for (j = 0; j < 15; j++){
382 if (column[j] == 1){
383 count++;
384 meany = meany + j;
385 }
386 }
387 if (count != 0){
388 meany = meany/count;
389 }
390 printf("meanx=\n", meanx, meany);
391 printf("meany=\n", meanx, meany);
392 getMovement();
```

A.8  test.c

Program to check if the kinect input works

#include <stdio.h>
#include <pthread.h>
#include <math.h>
#include <GL/glut.h>
#include <GL/glu.h>
#include <GL/glu.h>
#include <stdio.h>
#include <threadfunc.h>
#include <drawX.h>

void *glThreadFunc2(void *arg)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize(1280, 480);
    glutInitWindowPosition(0, 0);
    window3 = glutCreateWindow("LibFreenect");
    glutDisplayFunc(&DrawGLScene);
    glutIdleFunc(&DrawGLScene);
    glutReshapeFunc(&ReSizeGLScene);
    glutKeyboardFunc (&keyPressed);
    glutSpecialFunc(&specialKeyPressed);
    InitGL(1280, 480);
    glutMainLoop();
    return NULL;
}

int main(int argc, char **argv)
{
    int res;
    program = KINECTTEST;
    depth_mid = (uint8_t*)malloc(640*480*3);
    rgb_back = (uint8_t*)malloc(640*480*3);
    rgb_mid = (uint8_t*)malloc(640*480*3);
    rgb_front = (uint8_t*)malloc(640*480*3);

    printf("Kinect camera test\n");
    for (i = 0; i < 2048; i++) {
        float v = i / 2048.0;
        v = powf(v, 3)*6;
        *gamma[i] = v*6+256;
    }
    g_argc = argc;
    g_argv = argv;

    if (freenect_init(&ctx, NULL) < 0) {
        printf("freenect_init() failed\n");
        return 1;
    }
    freenect_setLogLevel(ctx, FREENECT_LOGDEBUG);
    nr_devices = freenect_numDevices(ctx);
    printf("Number of devices found: %d\n", nr_devices);
    user_device_number = 0;
    if (argc > 1) {
        user_device_number = atoi(argv[1]);
    }
    if (nr_devices < 1)
        return 1;
A.9 defines.h

Code file with all defines

```c
#define TRUE 1
#define FALSE 0
#define ESCAPE 27
#define PAGE_UP 73
#define PAGE_DOWN 81
#define UP_ARROW 72
#define DOWN_ARROW 80
#define LEFT_ARROW 75
#define RIGHT_ARROW 77
#define KEY_JEL 127
#define KEY_PAD_UP 56
#define KEY_PAD_DOWN 50
#define KEY_PAD_LEFT 52
#define KEY_PAD_RIGHT 54
#define KEY_PAD_MIDD 53
#define KEY_PAD_LEFT_UP 55
#define KEY_PAD_LEFT_DOW 49
#define KEY_PAD_RIGHT_UP 57
#define KEY_PAD_RIGHT_DOW 51
#define MAX_IMAGES 48
#define KEEPSPINNING 0
#define AUTOSWITCH 1
#define MANUALSWITCH 2
#define DEFAULTVIEW 4
#define KUBUS 0
#define GLOBES 1
#define KINECTTEST 2
#define DEBUG 0

// change DEBUG to 1 if you want to see the kinect feed in Glview
```

A.10 drawX.h

Code of drawX.h
A.11 imageload.h

Code of imageload.h

```c
#ifndef IMAGELOAD_H
#define IMAGELOAD_H
#include "libfreeect.h"

int window, window2, window3; // The numbers of our GLUT windows
extern GLfloat ymoon;
extern GLfloat xmoon;
extern GLfloat ymoonrot;
extern GLfloat xmoonrot;
extern GLfloat xrot; // X Rotation
extern GLfloat yrot; // Y Rotation
extern GLfloat zrot; // Z Rotation
extern GLfloat zrotmirror; // Z Rotation
extern GLfloat xspeed; // x rotation speed
extern GLfloat yspeed; // y rotation speed
extern GLfloat zspeed; // Z rotation speed
extern GLfloat zspeedmirror; // Z rotation speed
extern GLfloat z1; // depth into the screen.
extern GLfloat z2; // depth into the screen.
extern GLshort transparent;
extern GLshort moon;
extern int type; // 0 for animate, 1 for switching textures, 2 for manually change
extern int teller; // counter to control the pace of changing textures.
extern int teller2; // counter to control the pace of changing textures.
extern int last; // last shown was the earth
extern int last2; // last shown was the earth
extern int teller3; // for manually changing the planets a counter to keep track of which planet we look at.
extern int bmpx2; // start of texture in second window
extern int bmpx; // start of texture in first window
extern int cycle; // variable to keep track of which led color the kinect must show

GLUquadricObj *quadratic; // Storage For Our Quadratic Objects
GLUquadricObj *quadratic2; // Storage For Our Quadratic Objects
GLUquadricObj *quadraticmoon; // Storage For Our Quadratic Objects

// function prototypes
void DrawGLScene();
void DrawGLScene2();
void DrawGLScene3();
void DrawGLScene4();
void InitGL2(GLsizei Width, GLsizei Height);
void InitGL(GLsizei Width, GLsizei Height);
void ReSizeGLScene(GLsizei Width, GLsizei Height);
void ReSizeGLScene2(GLsizei Width, GLsizei Height);

#endif
```

```c
#include "libfreeect.h"

char *filenamearray[MAXIMAGES]; // array with imagelocations
GLuint texture[MAXIMAGES]; // Storage for texture.

/* Image type — contains height, width and data */
struct Image {
  unsigned long sizeX;
  unsigned long sizeY;
  char *data;
}
```
typedef struct Image Image;
/
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
∗
*
A.13  kinect.h

Code of kinect.h

```c
#define KINECT_H

int xlowerbound;
int xupperbound;
int ylowerbound;
int yupperbound;
int zlowerbound;
int zupperbound;

int loc[480][640];
int smallblocks[240][320];
int mediumblocks[120][160];
int largeblocks[60][80];
int bigblocks[30][40];
int hugoblocks[15][20];

int blob[15][15];

void kinectGetPosition();
void kinectGetState();
void kinectSetActiveArea(int xlow, int xhigh, int ylow, int yhigh, int zlow, int zhigh);
void getMovement();
void getBlobs();
void emptyarray();

#endif
```

A.14  libfreenect.h

Code of libfreenect.h

```c
/* This file is part of the OpenKinect Project. http://www.openkinect.org */
/* Copyright (c) 2010 individual OpenKinect contributors. See the CONTRIB file */
/* for details. */
/* This code is licensed to you under the terms of the Apache License, version */
/* 2.0, or, at your option, the terms of the GNU General Public License, */
/* version 2.0. See the APACHE20 and GPL2 files for the text of the licenses, */
/* or the following URLs: */
/* http://www.apache.org/licenses/LICENSE-2.0 */
/* http://www.gnu.org/licenses/gpl-2.0.txt */
/* If you redistribute this file in source form, modified or unmodified, you */
/* may: */
/* 1) Leave this header intact and distribute it under the same terms, */
/* 2) Delete the Apache 2.0 clause and accompany it with the GPL2 file, or */
/* 3) Delete the GPL 2.0 clause and accompany it with the APACHE20 file */
/* In all cases you must keep the copyright notice intact and include a copy */
/* of the CONTRIB file. */
/* Binary distributions must follow the binary distribution requirements of */
/* either License. */
/* */
#endif
```
#define LIBFREENECT_H

#include <stdint.h>
#include <GL/glut.h>
#include <GL/gl.h>
#include "defines.h"
#include "interact.h"
#include "drawX.h"
#include "imageload.h"
#include "kinect.h"
#include "threadfunc.h"

#endif _plusplus

extern "C" {
#endif

thread_t freenect_thread;
extern volatile int die;

// back: owned by libfreenect (implicit for depth)
// mid: owned by callbacks, "latest frame ready"
// front: owned by GL, "currently being drawn"
uint8_t depth_mid, *depth_front;
uint8_t *rgb_back, *rgb_mid, *rgb_front;

GLuint gl_depth_tex;
GLuint gl_rgb_tex;

uint16_t *luma[2048];

int usage;
char **argv;

int main() {

#ifdef _plusplus

pthread_t freenect_thread;

extern volatile int die;

// Enumeration of available resolutions.
// Not all available resolutions are actually supported for all video formats.
// For instance, SXGA mode: the format might not perfectly match resolutions.
// For instance, 640x480 for the IB camera.

typedef enum {
    FREENECT_RESOLUTION_LOW = 0, /**< VGA - 320x240 */
    FREENECT_RESOLUTION_MEDIUM = 1, /**< YGA - 640x480 */
    FREENECT_RESOLUTION_HIGH = 2, /**< SXGA - 1280x1024 */
    FREENECT_RESOLUTION_HRMMY = 2147483647, /**< Dummy value to force enum to be 32 bits wide */
} freenect_resolution;

typedef enum {
    FREENECT_VIDEO_RGB = 0, /**< Decompressed RGB mode (demosaicing done by libfreenect) */
    FREENECT_VIDEO_BAYER = 1, /**< Bayer compressed mode (raw information from camera) */
    FREENECT_VIDEO_YUV_RGB = 2, /**< 8-bit IR mode */
    FREENECT_VIDEO_YUV_RAW = 3, /**< 10-bit IR mode */
    FREENECT_VIDEO_YUV_HRMMY = 4, /**< 10-bit packed IR mode */
    FREENECT_VIDEO_YUV_YUVRaw = 5, /**< YUV RGB mode */
    FREENECT_VIDEO_YUV = 6, /**< YUV Raw mode */
    FREENECT_VIDEO_YUV_HRJMBMY = 2147483647, /**< Dummy value to force enum to be 32 bits wide */
} freenect_video_format;

typedef enum {
    FREENECT_DEPTH_11BIT = 0, /**< 11 bit depth information in one uint16_t/pixel */
    FREENECT_DEPTH_10BIT = 1, /**< 10 bit depth information in one uint16_t/pixel */
    FREENECT_DEPTH_8BIT_PACKED = 2, /**< 11 bit depth information */
    FREENECT_DEPTH_10BIT_PACKED = 3, /**< 10 bit depth information */
    FREENECT_DEPTH_8BIT_HRMMY = 2147483647, /**< Dummy value to force enum to be 32 bits wide */
} freenect_depth_format;

// Structure to give information about the width, height, bitrate, framerate, and buffer size of a frame in a particular mode, as

}
typedef struct {
    int32_t reserved;        //**< unique ID used internally. The meaning of values
    led_t status;            //**< Status indicates the state the device is in.
    int8_t width;            //**< Width of the frame, in pixels /
    int8_t height;           //**< Height of the frame, in pixels /
    int8_t data_bits_per_pixel;  //**< Number of bits of information needed for each
    int8_t padding_bits_per_pixel; //**< Number of bits of padding for alignment used for
    int8_t frame_rate;       //**< Approximate expected frame rate, in Hz /
    int8_t valid;            //**< If 0, this freenect_frame_mode is invalid and does
    int8_t value;            //**< Total buffer size in bytes to hold a single frame
    int8_t bytes;            //**< Total buffer size in bytes to hold a single frame
    int8_t pv_mode;          //**< Hue of the frame, in pixels /
    int8_t pv_depth;         //**< Total number of bytes needed to hold a single frame.
    int8_t pv_pixels;        //**< Width of the frame, in pixels /
    int8_t pv_height;        //**< Height of the frame, in pixels /
    int8_t pv_width;         //**< Number of bits of information needed for each
    int8_t pv_padding_bits_per_pixel; //**< Number of bits of padding for alignment used for
    int8_t pv_frame_rate;    //**< Approximate expected frame rate, in Hz /
    int8_t pv_valid;         //**< If 0, this freenect_frame_mode is invalid and does
    int8_t pv_value;         //**< Total buffer size in bytes to hold a single frame
    int8_t pv_bytes;         //**< Total buffer size in bytes to hold a single frame
    int8_t pv_pv_mode;       //**< Hue of the frame, in pixels /
    int8_t pv_pv_depth;      //**< Total number of bytes needed to hold a single frame.
    int8_t pv_pv_pixels;     //**< Width of the frame, in pixels /
    int8_t pv_pv_height;     //**< Height of the frame, in pixels /
    int8_t pv_pv_width;      //**< Number of bits of information needed for each
    int8_t pv_pv_padding_bits_per_pixel; //**< Number of bits of padding for alignment used for
    int8_t pv_pv_frame_rate; //**< Approximate expected frame rate, in Hz /
    int8_t pv_pv_valid;      //**< If 0, this freenect_frame_mode is invalid and does
    int8_t pv_pv_value;      //**< Total buffer size in bytes to hold a single frame
    int8_t pv_pv_bytes;      //**< Total buffer size in bytes to hold a single frame
    int8_t pv_pv_pv_mode;    //**< Hue of the frame, in pixels /
    int8_t pv_pv_pv_depth;   //**< Total number of bytes needed to hold a single frame.
    int8_t pv_pv_pv_pixels;  //**< Width of the frame, in pixels /
    int8_t pv_pv_pv_height;  //**< Height of the frame, in pixels /
    int8_t pv_pv_pv_width;   //**< Number of bits of information needed for each
    int8_t pv_pv_pv_padding_bits_per_pixel; //**< Number of bits of padding for alignment used for
    int8_t pv_pv_pv_frame_rate; //**< Approximate expected frame rate, in Hz /
    int8_t pv_pv_pv_valid;   //**< If 0, this freenect_frame_mode is invalid and does
    int8_t pv_pv_pv_value;   //**< Total buffer size in bytes to hold a single frame
    int8_t pv_pv_pv_bytes;   //**< Total buffer size in bytes to hold a single frame
} freenect_frame_mode;
typedef libusb_context freenect_usb_context; /**< Holds libusb-1.0 specific information */
typedef void (*freenect(void))(void);
#endif WIN32
// If Win32, export all functions for DLL usage
#ifdef WIN32
#define FREENECTAPI /**< DLLExport information for windows, set to nothing on other platforms */
#endif

#ifdef DLLExport information for windows, set to nothing on other platforms */
#else
// this is required when building from a Win32 port of gcc without being force to compile all of the library files (.c) with g++...
#endif

#define FREENECTAPI extern "C" __declspec(dllimport)
#endif

typedef void (*freenect_callback)(freenect_context cb);

// Enumeration of message logging levels
typedef enum {
    FREENECTLOG_FATAL = 0, /**< Log for crashing/non-recoverable errors */
    FREENECTLOG_ERROR, /**< Log for major errors */
    FREENECTLOG_WARNING, /**< Log for warning messages */
    FREENECTLOG_NOTICE, /**< Log for important messages */
    FREENECTLOG_INFO, /**< Log for normal messages */
    FREENECTLOG_DEBUG, /**< Log for useful development messages */
    FREENECTLOG_SPW, /**< Log for slightly less useful messages */
    FREENECTLOG_FLOOD, /**< Log EVERYTHING. May slow performance. */
} freenect_loglevel;

/* Initialize a freenect context and do any setup required for platform specific USB libraries.
 * @param ctx Address of pointer to freenect context struct to allocate and initialize.
 * @param usb_usb context to initialize. Can be NULL if not using multiple contexts.
 * @return 0 on success, < 0 on error.
 */
FREENECTAPI int freenect_init(freenect_context *ctx, freenect_usb_context *usb_ctx);

/**
 * Closes the device if it is open, and frees the context
 * @param ctx freenect context to close/free
 * @return 0 on success
 */
FREENECTAPI int freenect_shutdown(freenect_context *ctx);

/**
 * Typedef for logging callback functions
 */
typedef void (*freenect_logcb)(freenect_context *dev, freenect_loglevel level, const char *msg);

/**
 * Set the log level for the specified freenect context
 * @param ctx freenect context to set log level for
 * @param level log level to use (see freenect_loglevel enum)
 */
FREENECTAPI void freenect_setloglevel(freenect_context *ctx, freenect_loglevel level);

/**
 * Callback for log messages (i.e. for rerouting to a file instead of stdout)
 * @param ctx freenect context to set log callback for
 * @param cb callback function pointer
 */
FREENECTAPI void freenect_setlogcallback(freenect_context *ctx, freenect_logcb cb);

/* Calls the platform specific usb event processor */
@param c t x  Context to process events for
@return 0 on success, other values on error, platform/library dependent

FREENECTAPI int freeneck_process_events(freenect_context *ctx);

/**
 * Return the number of kinect devices currently connected to the system
 * @param c t x  Context to access device count through
 * @return Number of devices connected, < 0 on error
 */
FREENECTAPI int freeneck_num_devices(freenect_context *ctx);

/**
 * Opens a kinect device via a context. Index specifies the index of the device
 * on the current state of the bus. Bus resets may cause indexes to shift.
 * @param c t x  Context to open device through
 * @return 0 on success, < 0 on error
 */
FREENECTAPI int freeneck_open_device(freenect_context *ctx, freeneck_device *dev, int index);

/**
 * Closes a device that is currently open
 * @param d ev  Device to close
 * @return 0 on success
 */
FREENECTAPI int freeneck_close_device(freenect_device *dev);

/**
 * Set the device user data, for passing generic information into callbacks
 * @param d ev  Device to attach user data to
 * @return Pointer to user data
 */
FREENECTAPI void freeneck_set_user(freenect_device *dev, void *user);

/**
 * Retrieve the pointer to user data from the device structure
 * @param d ev  Device from which to get user data
 * @return Pointer to user data
 */
FREENECTAPI void *freeneck_get_user(freenect_device *dev);

// / Typedef for depth image received event callbacks
typedef void (*freeneck_depth_cb)(freenect_device *dev, void *depth, uint32_t timestamp)

// / Typedef for video image received event callbacks
typedef void (*freeneck_video_cb)(freenect_device *dev, void *video, uint32_t timestamp)

/**
 * Set callback for depth information received event
 * @param d ev  Device to set callback for
 * @param cb  Function pointer for processing depth information
 */
FREENECTAPI void freeneck_set_depth_callback(freenect_device *dev, freeneck_depth_cb cb)

/**
 * Set callback for video information received event
 */

FREENECTAPI void freesenect_video_callback (freenect_device *dev, freesenect_video cb cb) ;

FREENECTAPI int freesenect_set_video_buffer (freenect_device *dev, void *buf);

FREENECTAPI int freesenect_set_depth_buffer (freenect_device *dev, void *buf);

FREENECTAPI int freesenect_start_video (freenect_device *dev);

FREENECTAPI int freesenect_start_depth (freenect_device *dev);

FREENECTAPI int freesenect_stop_video (freenect_device *dev);

FREENECTAPI int freesenect_stop_depth (freenect_device *dev);

FREENECTAPI int freesenect_update_tilt_state (freenect_device *dev);
FREENECTAPI freenectGetTiltState(freenectDevice *dev);

FREENECTAPI double freenectGetTiltDegs(freenectDevice *dev, double angle);

FREENECTAPI int freenectSetTiltDegs(freenectDevice *dev, double angle);

FREENECTAPI int freenectSetLed(freenectDevice *dev, freenectLedOptions option);

FREENECTAPI void freenectGetMksAccel(freenectTiltState *state, double *x, double *y, double *z);

FREENECTAPI int freenectGetVideoModeCount();
# FREENECTAPI

## Convenience function to return a mode descriptor matching the specified resolution and video camera pixel format, if one exists.

```c
const freenect_frame_mode* freenect_find_video_mode(int res, int fmt);
```

## Returns 0 on success, < 0 if error

```c
int freenect_find_video_mode(freenect_device* dev, const freenect_frame_mode* mode);
```

## Get the number of depth camera modes supported by the driver. This includes both RGB and IR modes.

```c
int freenect_find_depth_mode_count();
```
FREENECTAPI const freenect_frame_mode freenect_get_current_depth_mode(freenect_device *dev);

/**
 * Convenience function to return a mode descriptor matching the
 * specified resolution and depth camera pixel format, if one exists.
 * @param res Resolution desired
 * @param fmt Pixel format desired
 * @return A freenect_frame_mode that matches the arguments specified, if such a valid
 * mode exists; otherwise, an invalid freenect_frame_mode.
 */
FREENECTAPI const freenect_frame_mode freenect_find_depth_mode(freenect_resolution res, freenect_depth_format fmt);

/**
 * Sets the current depth mode for the specified device. The mode
 * cannot be changed while streaming is active.
 * @param dev Device for which to set the depth mode
 * @param mode Frame mode to set
 * @return 0 on success, < 0 if error
 */
FREENECTAPI int freenect_set_depth_mode(freenect_device* dev, const freenect_frame_mode mode);

#endif
#endif

A.15 threadfunc.h

Code of threadfunc.h

#include "libfreenect.h"

extern pthread_mutex_t g_mutex;
extern pthread_cond_t g_frame_cond;
extern int got_rgb;
extern int got_depth;

void *g_threadfunc(void *arg);
void depth_cb(freenect_device *dev, void *rgb, uint32_t timestamp);
void rgb_cb(freenect_device *dev, void *depth, uint32_t timestamp);
B Appendix B

B.1 textures

Figure 11: Textures
Figure 12: Textures

(a) Sun
(b) Uranus
(c) Venus