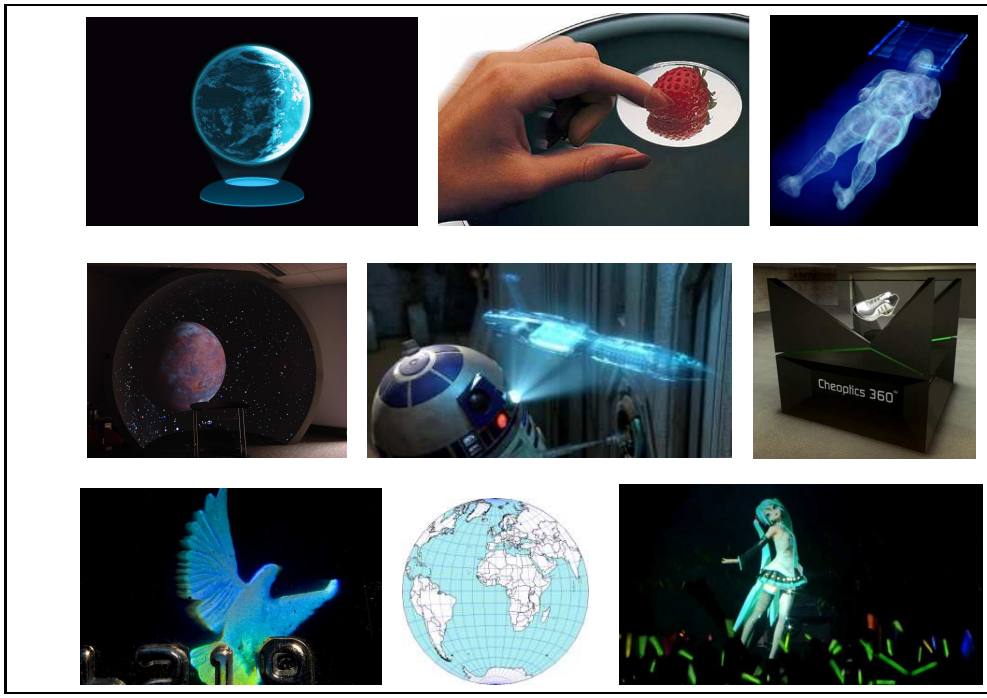
Holography and Kinect

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

Contents

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.

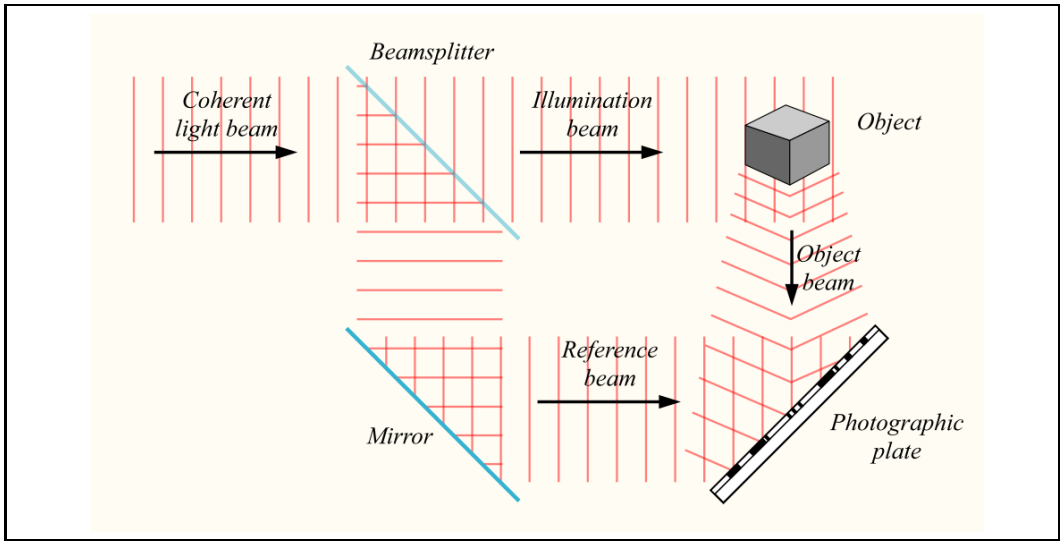
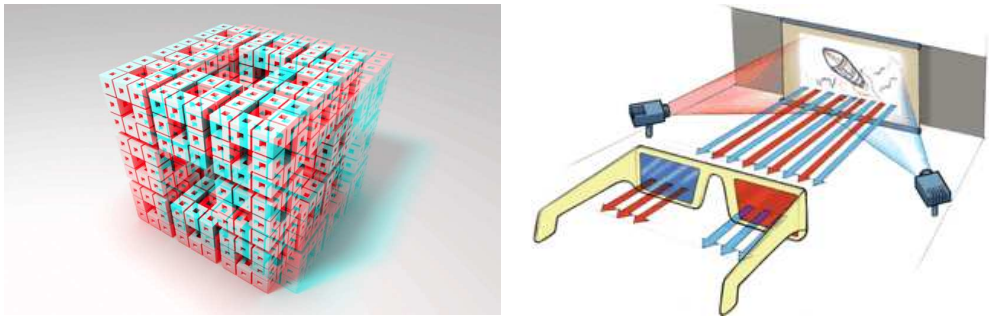


Figure 1: Recording the interference image

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.



(a) Anaglyph (b) Anaglyph technique

Figure 2: Anaglyph images

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

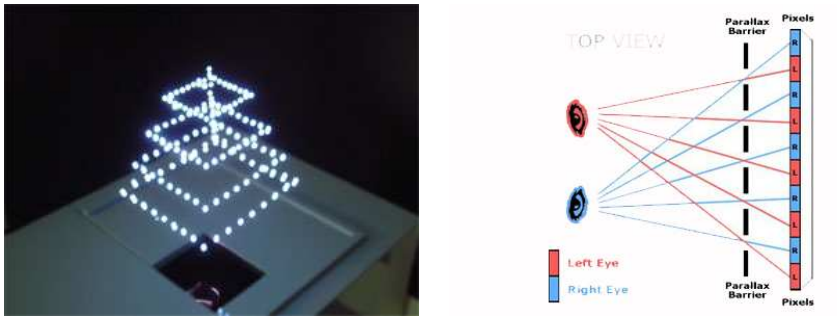
A glass pyramid with projections on all sides gives the illusion of an object inside the pyramid like the viZoo™ Cheoptics hologram [19] (see Figure 3a). An image is projected on top of the pyramid such that every side of the pyramid has a part of the object shown onto it. These images give the illusion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

Figure 3: Cheoptics 360 from ViZoo and Star Wars hologram R2D2

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.



(a) Plasmadots (b) Parallax technique

Figure 4: Plasma hologram and parallax technique

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.

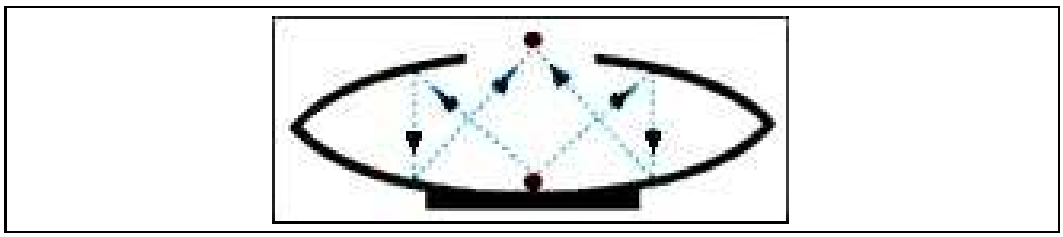


Figure 5: Concept optical illusion using a mirascope

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.

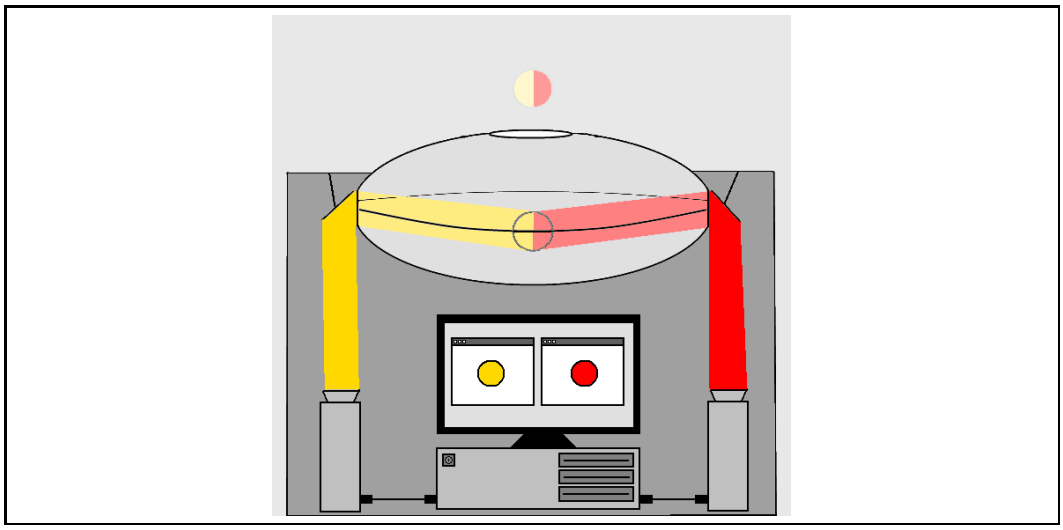


Figure 6: Concept optical illusion with beamer

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

Diameter	Height	Aperture	$F(x)$	second derivative
65	11	16	$F(x) = 0.010x^2$	$F''(x) = 0.02$
55.88	10.16	15.24	$F(x) = 0.0133x^2$	$F''(x) = 0.0266$
22.86	3.31	5.08	$F(x) = 0.0292x^2$	$F''(x) = 0.0584$
13.8	2.4	4.1	$F(x) = 0.053x^2$	$F''(x) = 0.106$

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

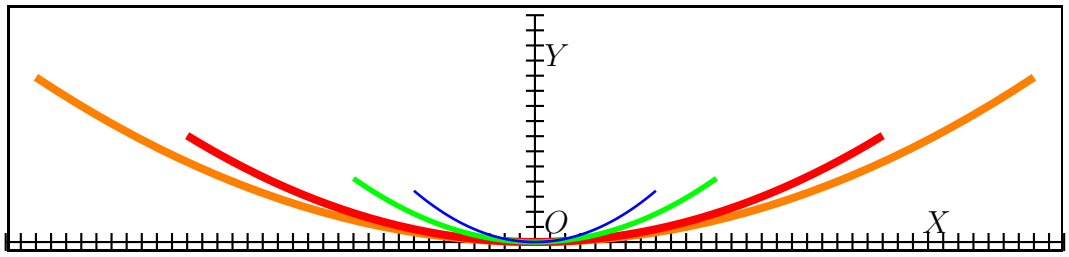


Figure 7: Graph showing different sizes of mirascopes

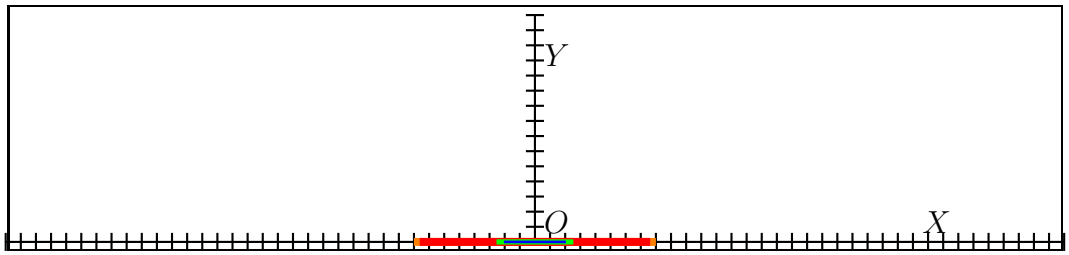


Figure 8: Graph showing apertures of different sizes of mirascopes

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.

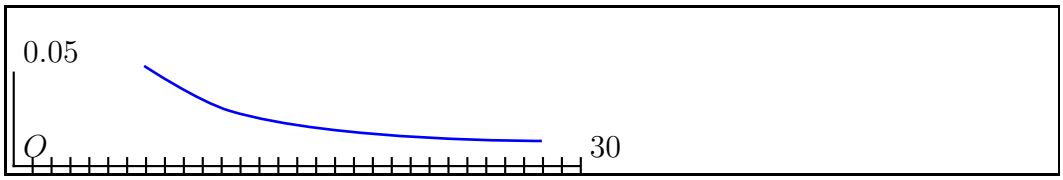


Figure 9: Graph showing relation between diameter/2 and factor a in $F(x) = ax^2 + bx + c$

Table 2: Aperture height relation

Total height	Aperture	$^{\text{Aperture}}/_{\text{Height}}$
22	16	0.7272
20.32	15.24	0.75
6.62	5.08	0.7673
4.8	4.1	0.8542

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].

(a) Correct viewpoint (b) Wrong viewpoint

Figure 10: Example of anamorphosis

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 ++

s: Kinect angle −−

x: Kinect angle default

]: Change color of the Kinects' LED

f: if in debug mode changes Kinect visual feed between 8Bit, RGB and YUV_RGB

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

[1] San Francisco Exploratorium Optical Illusion showing a Mirascope: `http://www.youtube.com/watch?v=XGIfdEjW-jM`, last visited 31 dec 2012

[2] J. P. Snyder, *Map Projections–A Working Manual*, Geological Survey, Series Number 1395, 1987, `http://en.wikipedia.org/wiki/Azimuthal_projection#Azimuthal_.28projections_onto_a_plane.29`, last visited 31 dec 2012

[3] Paul Bourke, *Using a spherical mirror for projection into immersive environments*, Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia. pp 281-284, 2006

[4] Rick Companje and Nico van Dijk and Hanco Hogenbierk and Danića Mast, *Globe4D, time-traveling with an interactive four-dimensional globe*, Proceeding SIGGRAPH '07 ACM SIGGRAPH 2007 emerging technologies Article No. 26 ACM New York, NY, USA 2007

[5] Sriya Adhya and John Noé, *A complete ray-trace analysis of the 'Mirage' toy*, ETOP-2007, Ottawa, 2007

[6] Optigone: `http://www.optigone.com/index.html`, last visited 31 dec 2012

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

[8] Jamie Shotton and Andrew Fitzgibbon and Mat Cook and Toby Sharp and Mark Finocchio and Richard Moore and Alex Kipman and Andrew Blake , *Real-Time Human Pose Recognition in Parts from Single Depth Images*, Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference, 20-25 June 2011, Page(s): 1297 - 1304, 2011

[9] Elvira Vreeswijk, *AnaMOVIEsis: An anamorphic 3D solution*, 2007

[10] Green Power Science: http://greenpowerscience.com, last visited 31 dec 2012

[11] Experimenten.nl: http://www.experimenten.nl/zilverspiegel.html, last visited 31 dec 2012

[12] Krylon Silver paint: http://krylon.com/products/looking_glass_mirror_like_paint, last visited 31 dec 2012

[13] Leandro Aparecido Nogueira de Paula and Pedro Raggio and André Koch Torres Assis *Uma Contribuio Construo de Espelhos Parablicos*, Caderno Brasileiro de Ensino de Fsica , v. 24, p. 338-352, 2007.

[14] K. Prazdny, *Detection of Binocular Disparities*, Journal: Biological Cybernetics, Volume 52, Issue 2, Page(s) 93 - 99, 1985

[15] Balázs Gulyás and Per E. Roland, *Binocular disparity discrimination in human cerebral cortex: Functional anatomy by positron emission tomography* Proceedings of the National Academy of Sciences of the United States of America, Volume: 91, Page(s) 1239 - 1243, Feb 1994

[16] Libfreenect: http://openkinect.org/wiki/Main_Page, last visited 31 dec 2012

[17] Angelgilding.com: http://angelgilding.com/DripSilverVideo.html, last visited 31 dec 2012

[18] Radu Bogdan Rusu and Steve Cousins, *3D is here: Point Cloud Library (PCL)*, Robotics and Automation (ICRA), 2011 IEEE International Conference 9-13 May 2011, Page(s) 1 - 4, 2011

[19] viZoo: http://www.vizoo.com/, last visited 31 dec 2012

[20] Microsoft Kinect SDK: http://www.microsoft.com/en-us/kinectforwindows/develop/resources.aspx, last visited 31 dec 2012

[21] Frederick Simeon and Kin Jue and Henry Wong and Frank Chiou, *3D Technologies ECE 477*, 2011

[22] Computer added animation / hologram:
`http://www.youtube.com/watch?v=RuUTkvfPpsA`, last visited 31 dec 2012

[23] Eric Dubois, *A projection method to generate anaglyph stereo images*, Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference, Volume 3, Page(s) 1661 - 1664, 2001

[24] True 3D display Aerial Burton:
`http://www.burton-jp.com/en/index.htm`, last visited 31 dec 2012

[25] Kibum Kim and John Bolton and Audrey Girouard and Jeremy Cooperstock and Roel Vertegaal, *TeleHuman: Effects of 3D Perspective on Gaze and Pose Estimation with a Life-size Cylindrical Telepresence Pod*, CHI '12 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Page(s) 2531 - 2540, 2012

[26] John Bolton and Peng Wang and Kibum Kim and Roel Vertegaal, *BodiPod: Interacting with 3D Human Anatomy via a 360 Cylindrical Display* Proceeding CHI EA '12 CHI '12 Extended Abstracts on Human Factors in Computing Systems, Page(s) 1039 - 1042, ACM New York, NY, USA 2012

A Appendix A

A.1 glview.c

Programs Main function for glview.

```
1  /*
2   * This file is part of the OpenKinect Project. http://www.openkinect.org
3   *
4   * Copyright (c) 2010 individual OpenKinect contributors. See the CONTRIB file
5   * for details.
6   *
7   * This code is licensed to you under the terms of the Apache License, version
8   * 2.0, or, at your option, the terms of the GNU General Public License,
9   * version 2.0. See the APACHE20 and GPL2 files for the text of the licenses,
10  * or the following URLs:
11  * http://www.apache.org/licenses/LICENSE-2.0
12  * http://www.gnu.org/licenses/gpl-2.0.txt
13  *
14  * If you redistribute this file in source form, modified or unmodified, you
15  * may:
16  *   1) Leave this header intact and distribute it under the same terms,
17  *      accompanying it with the APACHE20 and GPL20 files, or
18  *   2) Delete the Apache 2.0 clause and accompany it with the GPL2 file, or
19  *   3) Delete the GPL v2 clause and accompany it with the APACHE20 file
20  * In all cases you must keep the copyright notice intact and include a copy
21  * of the CONTRIB file.
22  *
23  * Binary distributions must follow the binary distribution requirements of
24  * either License.
25  */
26
27
28 #include <stdio.h>
29 #include <pthread.h>
30 #include <math.h>
31
32 #include "libfreenect.h"
33 #include "threadfunc.h"
34
35
36 int main(int argc, char **argv){
37  program = GLOBES;
38  FillArrayImageNames();
39  //set frame in which to recognize gestures
40  kinect_set_active_area(0,480,0,640,550,650);
41  update = 0;
42  meanxoud = 0;
43  meanyoud = 0;
44  int res;
45
46  depth_mid = (uint8_t*)malloc(640*480*3);
47  depth_front = (uint8_t*)malloc(640*480*3);
48  rgb_back = (uint8_t*)malloc(640*480*3);
49  rgb_mid = (uint8_t*)malloc(640*480*3);
50  rgb_front = (uint8_t*)malloc(640*480*3);
51
52  printf("Kinect camera test\n");
53
54  int i;
55  for (i=0; i<2048; i++) {
56   float v = i/2048.0;
57   v = powf(v, 3)* 6;
58   t_gamma[i] = v*6*256;
59  }
60
61  g_argc = argc;
62  g_argv = argv;
63
64  if (freenect_init(&f_ctx, NULL) < 0) {
65   printf("freenect_init() failed\n");
66   return 1;
67  }
68
```

```
69   freenect_set_log_level(f_ctx, FREENECT_LOG_DEBUG);
70
71   int nr_devices = freenect_num_devices (f_ctx);
72   printf ("Number of devices found: %d\n", nr_devices);
73
74   int user_device_number = 0;
75   if (argc > 1)
76    user_device_number = atoi(argv[1]);
77
78   if (nr_devices < 1)
79    return 1;
80
81   if (freenect_open_device(f_ctx, &f_dev, user_device_number) < 0) {
82    printf("Could not open device\n");
83    return 1;
84   }
85
86   res = pthread_create(&freenect_thread, NULL, freenect_threadfunc, NULL);
87   if (res) {
88    printf("pthread_create failed\n");
89    return 1;
90   }
91
92   // OS X requires GLUT to run on the main thread
93   gl_threadfunc(NULL);
94
95   return 0;
96  }
```

A.2 threadfunc.c

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

```
1  #include <assert.h>
2  #include <stdio.h>
3  #include <GL/glut.h>
4  #include <GL/gl.h>
5  #include <GL/glu.h>
6  #include <pthread.h>
7  #include "libfreenect.h"
8  #include "threadfunc.h"
9
10 pthread_mutex_t gl_backbuf_mutex = PTHREAD_MUTEX_INITIALIZER;
11 pthread_cond_t gl_frame_cond = PTHREAD_COND_INITIALIZER;
12
13
14 freenect_video_format requested_format = FREENECT_VIDEO_RGB;
15 freenect_video_format current_format = FREENECT_VIDEO_RGB;
16
17 int got_rgb = 0;
18 int got_depth = 0;
19
20 void *gl_threadfunc(void *arg){
21   glutInit(&g_argc, g_argv);
22   glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |  GLUT_DEPTH );
23
24   /* get a 640 x 480 window */
25   glutInitWindowSize(640, 480);
26   /* the window starts at the upper left corner of the screen */
27   glutInitWindowPosition(0, 0);
28   /* Open a window */
29   window = glutCreateWindow("2");  // first create the reacting window.
30   /* Register the function to do all our OpenGL drawing. */
31   glutDisplayFunc(&DrawGLScene2);
32   /* Even if there are no events, redraw our gl scene. */
33   glutIdleFunc(&DrawGLScene2);
34   /* Register the function called when our window is resized. */
35   glutReshapeFunc(&ReSizeGLScene2);
36   /* Register the function called when the keyboard is pressed. */
37   glutKeyboardFunc(&keyPressed);
38   /* Register the function called when mouse left is klicked */
39   glutMouseFunc(&MouseEvent);
```

```
40    /* Register the function called when special keys (arrows, page down, etc) are pressed
          . */
41    glutSpecialFunc(&specialKeyPressed);
42    InitGL2(640, 480);
43
44    /* the window starts at the upper left corner of the screen */
45    glutInitWindowPosition(640, 0);
46    /* Open a second window */
47    window2 = glutCreateWindow("1");
48     /* Even if there are no events, redraw our gl scene. */
49    glutDisplayFunc(&DrawGLScene4);
50      /* Even if there are no events, redraw our gl scene. */
51    glutIdleFunc(&DrawGLScene4);
52    /* Register the function called when our window is resized. */
53    glutReshapeFunc(&ReSizeGLScene2);
54    /* Register the function called when the keyboard is pressed. */
55    glutKeyboardFunc(&keyPressed);
56    /* Register the function called when mouse left is klicked */
57    glutMouseFunc(&MouseEvent);
58    /* Register the function called when special keys (arrows, page down, etc) are pressed
          . */
59    glutSpecialFunc(&specialKeyPressed);
60    glutMotionFunc(MouseMotion);
61    /* Initialize our window. */
62    InitGL2(640, 480);
63
64    if (DEBUG == 1){
65      glutInitWindowSize(1280, 480);
66      glutInitWindowPosition(0, 520);
67      window3 = glutCreateWindow("LibFreenect");
68      glutDisplayFunc(&DrawGLScene);
69      glutIdleFunc(&DrawGLScene);
70      glutReshapeFunc(&ReSizeGLScene);
71      /* Register the function called when the keyboard is pressed. */
72      glutKeyboardFunc(&keyPressed);
73      /* Register the function called when mouse left is klicked */
74      glutMouseFunc(&MouseEvent);
75      /* Register the function called when special keys (arrows, page down, etc) are
            pressed. */
76      glutSpecialFunc(&specialKeyPressed);
77      InitGL(1280, 480);
78    }
79    glutMainLoop();
80    return NULL;
81  }
82  //
83
84  void depth_cb(freenect_device *dev, void *v_depth, uint32_t timestamp)
85  {
86   int i,j;
87   uint16_t *depth = (uint16_t*)v_depth;
88   //save the depth map in a two dimensional array for that is easy to work with
89   for (i = 0; i < 480; i++){
90     for (j = 0; j < 640; j++){
91         location[i][j] = depth[i*640+j];
92     }
93   }
94
95   pthread_mutex_lock(&gl_backbuf_mutex);
96   for (i=0; i<640*480; i++) {
97    int pval = t_gamma[depth[i]];
98    int lb = pval & 0xff;
99    switch (pval>>8) {
100    case 0:
101         depth_mid[3*i+0] = 255;
102         depth_mid[3*i+1] = 255-lb;
103         depth_mid[3*i+2] = 255-lb;
104       break;
105    case 1:
106         depth_mid[3*i+0] = 255;
107         depth_mid[3*i+1] = lb;
108         depth_mid[3*i+2] = 0;
109       break;
110    case 2:
111         depth_mid[3*i+0] = 255-lb;
112         depth_mid[3*i+1] = 255;
113         depth_mid[3*i+2] = 0;
```

```
114       break;
115     case 3:
116         depth_mid[3*i+0] = 0;
117         depth_mid[3*i+1] = 255;
118         depth_mid[3*i+2] = lb;
119       break;
120     case 4:
121         depth_mid[3*i+0] = 0;
122         depth_mid[3*i+1] = 255-lb;
123         depth_mid[3*i+2] = 255;
124       break;
125     case 5:
126         depth_mid[3*i+0] = 0;
127         depth_mid[3*i+1] = 0;
128         depth_mid[3*i+2] = 255-lb;
129       break;
130     default:
131       depth_mid[3*i+0] = 0;
132       depth_mid[3*i+1] = 0;
133       depth_mid[3*i+2] = 0;
134       break;
135    }
136   }
137   got_depth++;
138   pthread_cond_signal(&gl_frame_cond);
139   pthread_mutex_unlock(&gl_backbuf_mutex);
140  }
141
142  void rgb_cb(freenect_device *dev, void *rgb, uint32_t timestamp)
143  {
144   pthread_mutex_lock(&gl_backbuf_mutex);
145   // swap buffers
146   assert (rgb_back == rgb);
147   rgb_back = rgb_mid;
148   freenect_set_video_buffer(dev, rgb_back);
149   rgb_mid = (uint8_t*)rgb;
150
151   got_rgb++;
152   pthread_cond_signal(&gl_frame_cond);
153   pthread_mutex_unlock(&gl_backbuf_mutex);
154  }
155
156  void *freenect_threadfunc(void *arg)
157  {
158   int accelCount = 0;
159
160   freenect_set_tilt_degs(f_dev,freenect_angle);
161   freenect_set_led(f_dev,LED_RED);
162   freenect_set_depth_callback(f_dev, depth_cb);
163   freenect_set_video_callback(f_dev, rgb_cb);
164   freenect_set_video_mode(f_dev, freenect_find_video_mode(FREENECT_RESOLUTION_MEDIUM,
          current_format));
165   freenect_set_depth_mode(f_dev, freenect_find_depth_mode(FREENECT_RESOLUTION_MEDIUM,
          FREENECT_DEPTH_11BIT));
166   freenect_set_video_buffer(f_dev, rgb_back);
167
168   freenect_start_depth(f_dev);
169   freenect_start_video(f_dev);
170
171   //printf("'w'-tilt up, 's'-level, 'x'-tilt down, '0'-'6'-select LED mode, 'f'-video
          format\n");
172
173   while (freenect_process_events(f_ctx) >= 0) {
174    //Throttle the text output
175    if (accelCount++ >= 2000)
176    {
177     accelCount = 0;
178     freenect_raw_tilt_state* state;
179     freenect_update_tilt_state(f_dev);
180     state = freenect_get_tilt_state(f_dev);
181     double dx,dy,dz;
182     freenect_get_mks_accel(state, &dx, &dy, &dz);
183     //printf("\r raw acceleration: %4d %4d %4d  mks acceleration: %4f %4f %4f", state->
            accelerometer_x, state->accelerometer_y, state->accelerometer_z, dx, dy, dz);
184     fflush(stdout);
185    }
186
```

```
187    if (requested_format != current_format) {
188     freenect_stop_video(f_dev);
189     freenect_set_video_mode(f_dev, freenect_find_video_mode(FREENECT_RESOLUTION_MEDIUM,
            requested_format));
190     freenect_start_video(f_dev);
191     current_format = requested_format;
192    }
193   }
194
195   //printf("\nshutting down streams...\n");
196
197   freenect_stop_depth(f_dev);
198   freenect_stop_video(f_dev);
199
200   freenect_close_device(f_dev);
201   freenect_shutdown(f_ctx);
202
203   //printf("-- done!\n");
204   return NULL;
205  }
```

A.3 drawX.c

This is the main draw code.

```
1  #include <stdio.h>
2  #include <GL/glut.h>
3  #include <GL/gl.h>
4  #include <GL/glu.h>
5  #include <pthread.h>
6  #include <math.h>
7  #include "libfreenect.h"
8  #include "drawX.h"
9  #include "threadfunc.h"
10
11 int overlayproject = 20;
12 int interval = 0;
13
14 GLboolean animate = GL_FALSE;
15 int type = 2;    // 0 for animate, 1 for switching textures, 2 for manually change
16 int teller = 700;  // counter to control the pace of changing textures.
17 int teller2 = 700;  // counter to control the pace of changing textures.
18 int last = 2;    // last shown was the earth
19 int last2 = 12;    // last shown was the earth
20 int teller3 = 0;
21 int planet = 2;    // for manually changing the planets a counter to keep track of which
       planet we look at.
22 int bmpx = 10;    // startnr of texture in second window
23 int bmpx2 = 0;    // startnr of texture in first window
24 int cycle = 0;    // variable to keep track of wich led color the kinect must show
25
26 GLfloat xmoon = 0.0f;
27 GLfloat ymoon = 0.0f;
28 GLfloat xmoonrot = 0.0f;
29 GLfloat ymoonrot = 0.0f;
30 GLfloat yrot = 0.0f;  // X Rotation
31 GLfloat xrot = 0.0f;  // Y Rotation
32 GLfloat zrot = 0.0f;  // Z Rotation
33 GLfloat zrotmirror = 0.0f; // Z Rotation mirrored
34 GLfloat xspeed = 0.0f;  // x rotation speed
35 GLfloat yspeed = 0.0f;  // y rotation speed
36 GLfloat zspeed = 0.0f;  // Z rotation speed
37 GLfloat zspeedmirror = 0.0f; // Z rotation speed mirrored
38 GLfloat z1=-3.8f;  // depth into the screen.
39 GLfloat z2=-3.8f;  // depth into the screen.
40 GLshort transparent = FALSE;
41 GLshort moon = FALSE;
42 GLfloat twoscreen = 1.95f; //shift to left or to the right to accomodate 2 monitor
       version
43 GLfloat correctie = 0.0f; //variable to correct viewpoint while lateral turning
44 int framecounter = 0;
45
46 //Light declarations
```

```
47 /* white ambient light at half intensity (rgba) */
48 GLfloat LightAmbient[] = { 1.5f, 1.5f, 1.5f, 1.0f };
49 /* super bright, full intensity diffuse light. */
50 GLfloat LightDiffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };
51 /* position of light (x, y, z, (position of light)) */
52 GLfloat LightPosition[] = { 1.0f, 1.0f, 2.0f, 1.0f };
53
54 void DrawGLScene2()
55 {
56     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  // Clear The Screen And The
          Depth Buffer
57     glLoadIdentity();     // Reset The View
58
59     yrot+=xspeed;                    // X Axis Rotation
60     xrot+=yspeed;                    // Y Axis Rotation
61     zrot+=zspeed;     // Z Axis Rotation
62     zrotmirror+=zspeedmirror;
63
64     if (yrot > 360.0f || yrot < -360.0f) yrot = 0.0f;
65     if (xrot > 360.0f || xrot < -360.0f) xrot = 0.0f;
66     if (zrot > 360.0f || zrot < -360.0f) zrot = 0.0f;
67     if (zrotmirror > 360.0f || zrotmirror < -360.0f) zrotmirror = 0.0f;
68
69     ymoonrot+=xmoon;
70     xmoonrot+=ymoon;
71
72     // slow down after a while.
73     if (xspeed > 0) xspeed = xspeed - 0.0001f;
74     else if (xspeed < 0) xspeed = xspeed + 0.0001f;
75     if (yspeed > 0) yspeed = yspeed - 0.0001f;
76     else if (yspeed < 0) yspeed = yspeed + 0.0001f;
77     if (zspeed > 0) zspeed = zspeed - 0.0001f;
78     else if (zspeed < 0) zspeed = zspeed + 0.0001f;
79     if (zspeedmirror > 0) zspeedmirror = zspeedmirror - 0.0001f;
80     else if (zspeedmirror < 0) zspeedmirror = zspeedmirror + 0.0001f;
81     if (xmoon > 0) xmoonrot = xmoonrot - 0.0001f;
82     else if (xmoon < 0) xmoon = xmoon + 0.0001f;
83     if (ymoon > 0) ymoon = ymoon - 0.0001f;
84     else if (ymoon < 0) ymoon = ymoon + 0.0001f;
85
86
87     bmpx2 = last;
88     //A switch to select what texture has to be shown on the solid sphere
89     switch (type){
90  case KEEPSPINNING: // animate
91       glBindTexture(GL_TEXTURE_2D, texture[2]);
92    break;
93  case AUTOSWITCH: //switching textures
94       if (teller2 == 0) teller2 = 700;
95       if (teller2 % 50 == 0){
96         teller2--;
97         last++;
98         if (last > 10) last = 0;
99       }
100      else teller2--;
101      glBindTexture(GL_TEXTURE_2D, texture[bmpx2]);
102   break;
103 case MANUALSWITCH: //manually switch textures
104   switch (planet) {
105     case 0:  //mercury
106  glBindTexture(GL_TEXTURE_2D, texture[0]);
107  last = 0;
108       break;
109     case 1:  //venus
110  glBindTexture(GL_TEXTURE_2D, texture[1]);
111  last = 1;
112       break;
113     case 2:  //earth
114  glBindTexture(GL_TEXTURE_2D, texture[2]);
115  last = 2;
116       break;
117     case 3:  //mars
118  glBindTexture(GL_TEXTURE_2D, texture[3]);
119  last = 3;
120       break;
121     case 4:  //jupiter
122  glBindTexture(GL_TEXTURE_2D, texture[4]);
```

```
123    last = 4;
124          break;
125        case 5:  //saturn
126    glBindTexture(GL_TEXTURE_2D, texture[5]);
127    last = 5;
128          break;
129        case 6:  //uranus
130    glBindTexture(GL_TEXTURE_2D, texture[6]);
131    last = 6;
132          break;
133        case 7:  //neptune
134    glBindTexture(GL_TEXTURE_2D, texture[7]);
135    last = 7;
136          break;
137        case 8:  //pluto
138    glBindTexture(GL_TEXTURE_2D, texture[8]);
139    last = 8;
140          break;
141        case 9:  //sun
142          glBindTexture(GL_TEXTURE_2D, texture[9]);
143            last = 9;
144          break;
145        case 10:  //moon
146          glBindTexture(GL_TEXTURE_2D, texture[46]);
147            last = 46;
148          break;
149        default:
150          glBindTexture(GL_TEXTURE_2D, texture[2]);
151          break;
152        }
153     break;
154   default:
155       glBindTexture(GL_TEXTURE_2D, texture[last]);    // choose the texture to use.
156     break;
157      }
158      //make sure globe fills the maximum amount of the screen.
159      //glTranslatef(-1.8f,0.0f,z1);              // move z units into the screen.
160      glTranslatef(0.0f,0.0f,z1);              // move z units into the screen.
161      glTranslatef(twoscreen,0.0f,0.0f);  // move twoscreen units to the side
162      GLfloat hulp = fmodf(yrot, 360.0f);
163
164      if (hulp != 0){
165        correctie = 1/hulp;
166        if (correctie > 1.0f || correctie < -1.0f){
167   correctie = 0.0f;
168        }
169        if (correctie > 0.042){
170   correctie = 0.042;
171        }
172        if (correctie < -0.042){
173   correctie = -0.042;
174        }
175      }
176
177      if (DEBUG && framecounter == 15){
178        printf("yrot = %f, xrot = %f, zrot = %f, correctie = %f\n", yrot, xrot, zrot,
               correctie);
179        framecounter = 0;
180      }
181      framecounter++;
182      //the rotation function regarding the speed
183      glRotatef(yrot,-1.0f,0.0f,-0.42f + correctie);  // Rotate On The X Axis
184      glRotatef(xrot,0.0f,1.0f,0.0f);     // Rotate On The Y Axis
185      glRotatef(zrot,-1.0f,1.0f,-0.42f + correctie);  // Rotate on the Z Axis
186      glRotatef(zrotmirror,1.0f,1.0f,0.42f + correctie); // Rotate on the Z mirror Axis
187
188      //turn globe into 'normal position'
189      glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
190      glRotatef(-45.0f, 0.0f, 0.0f, 1.0f);
191      gluSphere(quadratic,1.3f,32,32);                // Draw A Sphere
192
193      //check if overlayproject needs to be drawn
194      if (transparent) {
195        //enable alpha testing
196        glEnable(GL_ALPHA_TEST);
197        //enable blending
198        glEnable (GL_BLEND);
```

```
199         // set kind of blening
200         glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
201         //select texture for second bigger transparent sphere
202         if (animate){
203  if (overlayproject == 42) {
204    overlayproject = 20;
205  }
206  glBindTexture(GL_TEXTURE_2D, texture[overlayproject]);   // choose the texture to use.
207  if (interval > 10){
208    overlayproject++;
209    interval = 0;
210  }
211  interval++;
212         }
213         else {
214  glBindTexture(GL_TEXTURE_2D, texture[overlayproject]);   // choose the texture to use
215         }
216
217         //draw a second sphere on top of the first sphere
218         gluSphere(quadratic2,1.4f,32,32);
219         //disable alpha testing
220         glDisable(GL_ALPHA_TEST);
221         //disable blening
222         glDisable (GL_BLEND);
223      }
224
225      if (moon){
226        //glRotatef(-yrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
227        //glRotatef(-xrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
228        //glRotatef(-zrot,0.0f,0.0f,1.0f);  // Rotate on the Z Axis
229
230        glTranslatef(0.0f,3.0f,0.0f);            // move x units into the screen.
231        glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
232        //glRotatef(ymoonrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
233        //glRotatef(xmoonrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
234        glBindTexture(GL_TEXTURE_2D, texture[46]);
235
236        gluSphere(quadraticmoon,0.2f,32,32);          // Draw A moon
237        //glRotatef(-xmoonrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
238        //glRotatef(-ymoonrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
239        glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
240        glTranslatef(0.0,-3.0f,0.0f);            // move x units into the screen.
241      }
242      //set lighting
243      if (!light) {
244        glDisable(GL_LIGHTING);
245      } else {
246        glEnable(GL_LIGHTING);
247      }
248
249      glutPostRedisplay();
250      // since this is double buffered, swap the buffers to display what just got drawn.
251      glutSwapBuffers();
252  }
253
254  void DrawGLScene4()
255  {
256      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  // Clear The Screen And The
            Depth Buffer
257      glLoadIdentity();     // Reset The View
258
259      yrot+=xspeed;                    // X Axis Rotation
260      xrot+=yspeed;                    // Y Axis Rotation
261      zrot+=zspeed;     // Z Axis Rotation
262      zrotmirror+= zspeedmirror;
263
264      if (yrot > 360.0f || yrot < -360.0f) yrot = 0.0f;
265      if (xrot > 360.0f || xrot < -360.0f) xrot = 0.0f;
266      if (zrot > 360.0f || zrot < -360.0f) zrot = 0.0f;
267      if (zrotmirror > 360.0f || zrotmirror < -360.0f) zrotmirror = 0.0f;
268
269      ymoonrot+=xmoon;
270      xmoonrot+=ymoon;
271
272      // slow down after a while.
273      if (xspeed > 0) xspeed = xspeed - 0.0001f;
274      else if (xspeed < 0) xspeed = xspeed + 0.0001f;
```

```
275     if (yspeed > 0) yspeed = yspeed - 0.0001f;
276     else if (yspeed < 0) yspeed = yspeed + 0.0001f;
277     if (zspeed > 0) zspeed = zspeed - 0.0001f;
278     else if (zspeed < 0) zspeed = zspeed + 0.0001f;
279     if (zspeedmirror > 0) zspeedmirror = zspeedmirror - 0.0001f;
280     else if (zspeedmirror < 0) zspeedmirror = zspeedmirror + 0.0001f;
281     if (xmoon > 0) xmoonrot = xmoonrot - 0.0001f;
282     else if (xmoon < 0) xmoon = xmoon + 0.0001f;
283     if (ymoon > 0) ymoon = ymoon - 0.0001f;
284     else if (ymoon < 0) ymoon = ymoon + 0.0001f;
285
286     bmpx2 = last;
287     //A switch to select what texture has to be shown on the solid sphere
288     switch (type){
289   case KEEPSPINNING: // animate
290       glBindTexture(GL_TEXTURE_2D, texture[2]);
291     break;
292   case AUTOSWITCH: //switching textures
293       if (teller2 == 0) teller2 = 700;
294       if (teller2 % 50 == 0){
295         teller2--;
296         last++;
297         if (last > 10) last = 0;
298       }
299       else teller2--;
300       glBindTexture(GL_TEXTURE_2D, texture[bmpx2]);
301     break;
302   case MANUALSWITCH: //manually switch textures
303     switch (planet) {
304       case 0:  //mercury
305    glBindTexture(GL_TEXTURE_2D, texture[0]);
306    last = 0;
307         break;
308       case 1:  //venus
309    glBindTexture(GL_TEXTURE_2D, texture[1]);
310    last = 1;
311         break;
312       case 2:  //earth
313    glBindTexture(GL_TEXTURE_2D, texture[2]);
314    last = 2;
315         break;
316       case 3:  //mars
317    glBindTexture(GL_TEXTURE_2D, texture[3]);
318    last = 3;
319         break;
320       case 4:  //jupiter
321    glBindTexture(GL_TEXTURE_2D, texture[4]);
322    last = 4;
323         break;
324       case 5:  //saturn
325    glBindTexture(GL_TEXTURE_2D, texture[5]);
326    last = 5;
327         break;
328       case 6:  //uranus
329    glBindTexture(GL_TEXTURE_2D, texture[6]);
330    last = 6;
331         break;
332       case 7:  //neptune
333    glBindTexture(GL_TEXTURE_2D, texture[7]);
334    last = 7;
335         break;
336       case 8:  //pluto
337    glBindTexture(GL_TEXTURE_2D, texture[8]);
338    last = 8;
339         break;
340       case 9:  //sun
341         glBindTexture(GL_TEXTURE_2D, texture[9]);
342           last = 9;
343         break;
344       case 10:  //moon
345         glBindTexture(GL_TEXTURE_2D, texture[46]);
346           last = 46;
347         break;
348       default:
349         glBindTexture(GL_TEXTURE_2D, texture[2]);
350         break;
351       }
```

```
352     break;
353   default:
354       glBindTexture(GL_TEXTURE_2D, texture[last]);    // choose the texture to use.
355     break;
356      }
357      //make sure globe fills the maximum amount of the screen.
358      //glTranslatef(1.8f,0.0f,z1);              // move z units into the screen.
359      glTranslatef(0.0f,0.0f,z1);              // move z units into the screen.
360      glTranslatef(-twoscreen,0.0f,0.0f); // move twoscreen units to the side
361
362
363      //the rotation function regarding the speed
364      glRotatef(yrot,-1.0f,0.0f,0.42f+ correctie);  // Rotate On The X Axis
365      glRotatef(xrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
366      glRotatef(zrot,-1.0f,1.0f,0.42f + correctie);  // Rotate on the Z Axis
367      glRotatef(zrotmirror,1.0f,1.0f,-0.42f + correctie); // Rotate on the Z mirror Axis
368
369      //turn globe into 'normal position'
370      glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
371
372      //glRotatef(-180.0f, 0.0f, 0.0f, 1.0f);
373      gluSphere(quadratic,1.3f,32,32);                  // Draw A Sphere
374
375      //check if overlayproject needs to be drawn
376      if (transparent) {
377        //enable alpha testing
378        glEnable(GL_ALPHA_TEST);
379        //enable blending
380        glEnable (GL_BLEND);
381        // set kind of blening
382        glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
383        //select texture for second bigger transparent sphere
384        if (animate){
385   if (overlayproject == 42) overlayproject = 20;
386   glBindTexture(GL_TEXTURE_2D, texture[overlayproject]);    // choose the texture to use.
387   if (interval > 10){
388     overlayproject++;
389     interval = 0;
390   }
391        }
392        else {
393   glBindTexture(GL_TEXTURE_2D, texture[overlayproject]);    // choose the texture to use
394        }
395        interval++;
396        //draw a second sphere on top of the first sphere
397        gluSphere(quadratic2,1.4f,32,32);
398        //disable alpha testing
399        glDisable(GL_ALPHA_TEST);
400        //disable blening
401        glDisable (GL_BLEND);
402      }
403
404      if (moon){
405        //glRotatef(-yrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
406        //glRotatef(-xrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
407        //glRotatef(-zrot,0.0f,0.0f,1.0f);  // Rotate on the Z Axis
408
409        glTranslatef(0.0f,3.0f,0.0f);             // move x units into the screen.
410        glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
411        //glRotatef(ymoonrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
412        //glRotatef(xmoonrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
413        glBindTexture(GL_TEXTURE_2D, texture[47]);
414
415        gluSphere(quadraticmoon,0.2f,32,32);          // Draw A moon
416        //glRotatef(-xmoonrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
417        //glRotatef(-ymoonrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
418        glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
419        glTranslatef(0.0,-3.0f,0.0f);             // move x units into the screen.
420      }
421
422      //set lighting
423      if (!light) {
424        glDisable(GL_LIGHTING);
425      } else {
426        glEnable(GL_LIGHTING);
427      }
428
```

```
429     glutPostRedisplay();
430     // since this is double buffered, swap the buffers to display what just got drawn.
431     glutSwapBuffers();
432 }
433
434 void DrawGLScene3()
435 {
436     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  // Clear The Screen And The
          Depth Buffer
437     glLoadIdentity();     // Reset The View
438
439     yrot+=xspeed;                    // X Axis Rotation
440     xrot+=yspeed;                    // Y Axis Rotation
441     zrot+=zspeed;     // Z Axis Rotation
442     xmoonrot+=xmoon;
443     ymoonrot+=ymoon;
444
445     // slow down after a while.
446     if (xspeed > 0) xspeed = xspeed - 0.0001f;
447     else if (xspeed < 0) xspeed = xspeed + 0.0001f;
448     if (yspeed > 0) yspeed = yspeed - 0.0001f;
449     else if (yspeed < 0) yspeed = yspeed + 0.0001f;
450     if (zspeed > 0) zspeed = zspeed - 0.0001f;
451     else if (zspeed < 0) zspeed = zspeed + 0.0001f;
452     if (xmoonrot > 0) xmoonrot = xmoonrot - 0.0001f;
453     else if (xmoonrot < 0) xmoonrot = xmoonrot + 0.0001f;
454     if (ymoonrot > 0) ymoonrot = ymoonrot - 0.0001f;
455     else if (ymoonrot< 0) ymoonrot = ymoonrot + 0.0001f;
456
457     bmpx = last2;
458     switch (type){
459       case KEEPSPINNING: //animate
460    glBindTexture(GL_TEXTURE_2D, texture[12]);
461  break;
462       case AUTOSWITCH: //switchingtextures
463    if (teller == 0) teller = 700;
464    if (teller % 100 == 0){
465      teller--;
466      last2++;
467      if (last2 > 19) last2 = 10;
468    }
469    else teller--;
470    glBindTexture(GL_TEXTURE_2D, texture[bmpx]);
471  break;
472       case MANUALSWITCH: //manually change textures
473    switch (planet) {
474      case 0:  //mercury
475   glBindTexture(GL_TEXTURE_2D, texture[10]);
476   last2 = 10;
477        break;
478      case 1:  //venus
479          glBindTexture(GL_TEXTURE_2D, texture[11]);
480   last2 = 11;
481        break;
482      case 2:  //earth
483          glBindTexture(GL_TEXTURE_2D, texture[12]);
484   last2 = 12;
485        break;
486      case 3:  //mars
487          glBindTexture(GL_TEXTURE_2D, texture[13]);
488   last2 = 13;
489        break;
490      case 4:  //jupiter
491          glBindTexture(GL_TEXTURE_2D, texture[14]);
492   last2 = 14;
493        break;
494      case 5:  //saturn
495          glBindTexture(GL_TEXTURE_2D, texture[15]);
496   last2 = 15;
497        break;
498      case 6:  //uranus
499          glBindTexture(GL_TEXTURE_2D, texture[16]);
500   last2 = 16;
501        break;
502      case 7:  //neptune
503          glBindTexture(GL_TEXTURE_2D, texture[17]);
504   last2 = 17;
```

```
505         break;
506       case 8:  //pluto
507           glBindTexture(GL_TEXTURE_2D, texture[18]);
508   last2 = 18;
509         break;
510       case 9:  //sun
511           glBindTexture(GL_TEXTURE_2D, texture[19]);
512   last2 = 19;
513         break;
514       default:
515           glBindTexture(GL_TEXTURE_2D, texture[12]);
516         break;
517          }
518          break;
519
520       default:
521     glBindTexture(GL_TEXTURE_2D, texture[last2]);    // choose the texture to use.
522  break;
523     //manually change textures
524     }
525
526     //glTranslatef(1.8f,0.0f,z2);            // move z units into the screen, and 1.8
          units to the side.
527     glTranslatef(0.0f,0.0f,z2);            // move z units into the screen, and 1.8 units
          to the side.
528     glRotatef(-yrot,1.0f,0.0f,1.0f);  // Rotate On The X Axis
529     glRotatef(xrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
530     glRotatef(-zrot,0.0f,0.0f,1.0f);  // Rotate on the Z Axis
531
532     glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
533     gluSphere(quadratic,1.3f,32,32);                  // Draw A Sphere
534
535     // first check if needs to be transparent
536     if (transparent) {
537       //enable alpha testing
538       glEnable(GL_ALPHA_TEST);
539       //enable the blend function
540       glEnable (GL_BLEND);
541       // set blending function
542       glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
543       //select texture for second bigger transparent sphere
544       if (animate){
545  if (overlayproject == 42) overlayproject = 20;
546  glBindTexture(GL_TEXTURE_2D, texture[overlayproject]);    // choose the texture to use.
547  overlayproject++;
548       }
549       else {
550  glBindTexture(GL_TEXTURE_2D, texture[43]);    // choose the texture to use
551       }
552       //draw a second sphere on top of the first sphere
553       gluSphere(quadratic2,1.4f,32,32);
554       //disable alpha testing
555       glDisable(GL_ALPHA_TEST);
556       //disable blending
557       glDisable (GL_BLEND);
558     }
559
560     if (moon){
561       glRotatef(yrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
562       glRotatef(-xrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
563       glRotatef(zrot,0.0f,0.0f,1.0f);  // Rotate on the Z Axis
564
565       glTranslatef(0.0f,-2.0f,0.0f);            // move x units into the screen.
566       glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
567       glRotatef(ymoonrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
568       glRotatef(-xmoonrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
569       glBindTexture(GL_TEXTURE_2D, texture[9]);
570
571       gluSphere(quadraticmoon,0.2f,32,32);          // Draw A moon
572       glRotatef(-xmoonrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
573       glRotatef(ymoonrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
574       glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
575       glTranslatef(0.0,2.0f,0.0f);            // move x units into the screen.
576     }
577
578     //set lighting
579     if (!light) {
```

```
580         glDisable(GL_LIGHTING);
581       } else {
582         glEnable(GL_LIGHTING);
583       }
584
585       glutPostRedisplay();
586       // since this is double buffered, swap the buffers to display what just got drawn.
587       glutSwapBuffers();
588
589 }
590
591 void DrawGLScene()
592 {
593  pthread_mutex_lock(&gl_backbuf_mutex);
594
595  // When using YUV_RGB mode, RGB frames only arrive at 15Hz, so we shouldn't force them
       to draw in lock-step.
596  // However, this is CPU/GPU intensive when we are receiving frames in lockstep.
597  if (current_format == FREENECT_VIDEO_YUV_RGB) {
598   while (!got_depth && !got_rgb) {
599    pthread_cond_wait(&gl_frame_cond, &gl_backbuf_mutex);
600   }
601  } else {
602   while ((!got_depth || !got_rgb) && requested_format != current_format) {
603    pthread_cond_wait(&gl_frame_cond, &gl_backbuf_mutex);
604   }
605  }
606
607  if (requested_format != current_format) {
608   pthread_mutex_unlock(&gl_backbuf_mutex);
609   return;
610  }
611  //only get blobs every 2 updates
612  //if (update == 0){
613    kinect_get_position();
614
615   // update = 1;
616  //}
617  //update --;
618
619  uint8_t *tmp;
620
621  if (got_depth) {
622   tmp = depth_front;
623   depth_front = depth_mid;
624   depth_mid = tmp;
625   got_depth = 0;
626  }
627  if (got_rgb) {
628   tmp = rgb_front;
629   rgb_front = rgb_mid;
630   rgb_mid = tmp;
631   got_rgb = 0;
632  }
633
634  pthread_mutex_unlock(&gl_backbuf_mutex);
635
636  //getMovement(nrofblobs, biggest);
637
638  glBindTexture(GL_TEXTURE_2D, gl_depth_tex);
639  glTexImage2D(GL_TEXTURE_2D, 0, 3, 640, 480, 0, GL_RGB, GL_UNSIGNED_BYTE, depth_front);
640
641  glBegin(GL_TRIANGLE_FAN);
642  glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
643  glTexCoord2f(0, 0); glVertex3f(0,0,0);
644  glTexCoord2f(1, 0); glVertex3f(640,0,0);
645  glTexCoord2f(1, 1); glVertex3f(640,480,0);
646  glTexCoord2f(0, 1); glVertex3f(0,480,0);
647  glEnd();
648
649  glBindTexture(GL_TEXTURE_2D, gl_rgb_tex);
650  if (current_format == FREENECT_VIDEO_RGB || current_format == FREENECT_VIDEO_YUV_RGB)
651   glTexImage2D(GL_TEXTURE_2D, 0, 3, 640, 480, 0, GL_RGB, GL_UNSIGNED_BYTE, rgb_front);
652  else
653   glTexImage2D(GL_TEXTURE_2D, 0, 1, 640, 480, 0, GL_LUMINANCE, GL_UNSIGNED_BYTE,
         rgb_front+640*4);
654
```

```
655  glBegin(GL_TRIANGLE_FAN);
656  glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
657  glTexCoord2f(0, 0); glVertex3f(640,0,0);
658  glTexCoord2f(1, 0); glVertex3f(1280,0,0);
659  glTexCoord2f(1, 1); glVertex3f(1280,480,0);
660  glTexCoord2f(0, 1); glVertex3f(640,480,0);
661  glEnd();
662
663  glutSwapBuffers();
664 }
665
666 /* The function called when our window is resized  */
667 void ReSizeGLScene2(GLsizei Width, GLsizei Height)
668 {
669     if (Height==0){  // Prevent A Divide By Zero If The Window Is Too Small
670  Height=1;
671     }
672
673     glViewport(0, 0, Width, Height); // Reset The Current Viewport And Perspective
          Transformation
674
675     glMatrixMode(GL_PROJECTION);
676     glLoadIdentity();
677
678     gluPerspective(45.0f, (GLfloat)Width / (GLfloat)Height, 0.1f, 100.0f);
679     glMatrixMode(GL_MODELVIEW);
680 }
681
682 void ReSizeGLScene(int Width, int Height){
683    if (Height==0){  // Prevent A Divide By Zero If The Window Is Too Small
684  Height=1;
685     }
686   glViewport(0,0,Width,Height);
687   glMatrixMode(GL_PROJECTION);
688   glLoadIdentity();
689   glOrtho (0, 1280, 480, 0, -1.0f, 1.0f);
690   glMatrixMode(GL_MODELVIEW);
691 }
692
693 void InitGL(int Width, int Height)
694 {
695  glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
696  glClearDepth(1.0);
697  glDepthFunc(GL_LESS);
698  glDepthMask(GL_FALSE);
699  glDisable(GL_DEPTH_TEST);
700  glDisable(GL_BLEND);
701  glDisable(GL_ALPHA_TEST);
702  glEnable(GL_TEXTURE_2D);
703  glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
704  glShadeModel(GL_FLAT);
705
706  glGenTextures(1, &gl_depth_tex);
707  glBindTexture(GL_TEXTURE_2D, gl_depth_tex);
708  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
709  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
710
711  glGenTextures(1, &gl_rgb_tex);
712  glBindTexture(GL_TEXTURE_2D, gl_rgb_tex);
713  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
714  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
715
716  ReSizeGLScene(Width, Height);
717 }
718
719 /* A general OpenGL initialization function.  Sets all of the initial parameters. */
720 void InitGL2(GLsizei Width, GLsizei Height) // We call this right after our OpenGL
       window is created.
721 {
722     LoadGLTextures();     // Load the textures
723     glEnable(GL_TEXTURE_2D);    // Enable texture mapping
724
725     glClearColor(0.0f, 0.0f, 0.0f, 0.0f); // This Will Clear The Background Color To
          Black
726     glClearDepth(1.0);     // Enables Clearing Of The Depth Buffer
727     glDepthFunc(GL_LESS);    // The Type Of Depth Test To Do
728     glEnable(GL_DEPTH_TEST);    // Enables Depth Testing
```

```
729     glShadeModel(GL_SMOOTH);    // Enables Smooth Color Shading
730     glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
731     glMatrixMode(GL_PROJECTION);
732     glLoadIdentity();     // Reset The Projection Matrix
733
734     gluPerspective(45.0f,(GLfloat)Width/(GLfloat)Height,0.1f,100.0f); // Calculate The
            Aspect Ratio Of The Window
735
736     glMatrixMode(GL_MODELVIEW);
737
738     // set up light number 1.
739     glLightfv(GL_LIGHT1, GL_AMBIENT, LightAmbient);    // add lighting. (ambient)
740     glLightfv(GL_LIGHT1, GL_DIFFUSE, LightDiffuse);    // add lighting. (diffuse).
741     glLightfv(GL_LIGHT1, GL_POSITION,LightPosition);   // set light position.
742     glEnable(GL_LIGHT1);                               // turn light 1 on.
743
744     /* setup blending */
745     glColor4f(1.0f, 1.0f, 1.0f, 0.25);
746     quadratic=gluNewQuadric();                  // Create A Pointer To The Quadric
            Object ( NEW )
747     quadratic2=gluNewQuadric();                  // Create A Pointer To The Quadric
            Object ( NEW )
748     quadraticmoon=gluNewQuadric();  //moon
749
750     // Can also use GLU_NONE, GLU_FLAT
751     gluQuadricNormals(quadratic, GLU_SMOOTH);    // Create Smooth Normals
752     gluQuadricTexture(quadratic, GL_TRUE);       // Create Texture Coords ( NEW )
753
754     gluQuadricNormals(quadratic2, GLU_SMOOTH);    // Create Smooth Normals
755     gluQuadricTexture(quadratic2, GL_TRUE);       // Create Texture Coords ( NEW )
756
757     gluQuadricNormals(quadraticmoon, GLU_SMOOTH);    // Create Smooth Normals
758     gluQuadricTexture(quadraticmoon, GL_TRUE);       // Create Texture Coords ( NEW )
759 }
```

A.4 imageload.c

load the textures into the program.

```
1
2 #include <stdio.h>
3
4 #include "imageload.h"
5 #include "libfreenect.h"
6
7 /*
8  * Function Fill Array Image Names, fills an array with all the images that are used
9  */
10 void FillArrayImageNames(){
11   filenamearray[0] = "./bin/Data/texture/mercury.bmp";
12   filenamearray[1] = "./bin/Data/texture/venus.bmp";
13   filenamearray[2] = "./bin/Data/texture/earth.bmp";
14   filenamearray[3] = "./bin/Data/texture/mars.bmp";
15   filenamearray[4] = "./bin/Data/texture/jupiter.bmp";
16   filenamearray[5] = "./bin/Data/texture/saturn.bmp";
17   filenamearray[6] = "./bin/Data/texture/uranus.bmp";
18   filenamearray[7] = "./bin/Data/texture/neptune.bmp";
19   filenamearray[8] = "./bin/Data/texture/pluto.bmp";
20   filenamearray[9] = "./bin/Data/texture/sun.bmp";
21   filenamearray[10] = "./bin/Data/texture/mercury2.bmp";
22   filenamearray[11] = "./bin/Data/texture/venus2.bmp";
23   filenamearray[12] = "./bin/Data/texture/earth2.bmp";
24   filenamearray[13] = "./bin/Data/texture/mars2.bmp";
25   filenamearray[14] = "./bin/Data/texture/jupiter2.bmp";
26   filenamearray[15] = "./bin/Data/texture/saturn2.bmp";
27   filenamearray[16] = "./bin/Data/texture/uranus2.bmp";
28   filenamearray[17] = "./bin/Data/texture/neptune2.bmp";
29   filenamearray[18] = "./bin/Data/texture/pluto2.bmp";
30   filenamearray[19] = "./bin/Data/texture/sun2.bmp";
31   filenamearray[20] = "./bin/Data/texture/overlay/1.bmp";
32   filenamearray[21] = "./bin/Data/texture/overlay/2.bmp";
33   filenamearray[22] = "./bin/Data/texture/overlay/3.bmp";
```

```
34    filenamearray[23] = "./bin/Data/texture/overlay/4.bmp";
35    filenamearray[24] = "./bin/Data/texture/overlay/5.bmp";
36    filenamearray[25] = "./bin/Data/texture/overlay/6.bmp";
37    filenamearray[26] = "./bin/Data/texture/overlay/7.bmp";
38    filenamearray[27] = "./bin/Data/texture/overlay/8.bmp";
39    filenamearray[28] = "./bin/Data/texture/overlay/9.bmp";
40    filenamearray[29] = "./bin/Data/texture/overlay/10.bmp";
41    filenamearray[30] = "./bin/Data/texture/overlay/11.bmp";
42    filenamearray[31] = "./bin/Data/texture/overlay/12.bmp";
43    filenamearray[32] = "./bin/Data/texture/overlay/13.bmp";
44    filenamearray[33] = "./bin/Data/texture/overlay/14.bmp";
45    filenamearray[34] = "./bin/Data/texture/overlay/15.bmp";
46    filenamearray[35] = "./bin/Data/texture/overlay/16.bmp";
47    filenamearray[36] = "./bin/Data/texture/overlay/17.bmp";
48    filenamearray[37] = "./bin/Data/texture/overlay/18.bmp";
49    filenamearray[38] = "./bin/Data/texture/overlay/19.bmp";
50    filenamearray[39] = "./bin/Data/texture/overlay/20.bmp";
51    filenamearray[40] = "./bin/Data/texture/overlay/21.bmp";
52    filenamearray[41] = "./bin/Data/texture/overlay/22.bmp";
53    filenamearray[42] = "./bin/Data/texture/overlay/23.bmp";
54    filenamearray[43] = "./bin/Data/texture/overlay/test1.bmp";
55    filenamearray[44] = "./bin/Data/texture/overlay/test2.bmp";
56    filenamearray[45] = "./bin/Data/texture/old/crate.bmp";
57    filenamearray[46] = "./bin/Data/texture/moon2.bmp";
58    filenamearray[47] = "./bin/Data/texture/moon.bmp";
59  }
60
61  /*
62   *  Load texture into memory
63   *  LoadGLTextures runs throught the filenamearray and allocates memory
64   *  for all the images. And than makes textures out of them
65   */
66  void LoadGLTextures() {
67    int x = 0;
68    // Allocate space for texture
69    while (x < MAX_IMAGES){
70      image[x] = (Image *) malloc(sizeof(Image));
71      if (image[x] == NULL) {
72     printf("Error allocating space for image %d \n", x);
73     exit(0);
74      }
75      //printf("trying to allocate space for %s \n", filenamearray[x]);
76      if (!ImageLoad(filenamearray[x], image[x])) {
77       exit(1);
78      }
79      // create Texture
80      glGenTextures(3, &texture[x]);
81      // texture 2 (linear scaling)
82      glBindTexture(GL_TEXTURE_2D, texture[x]);   // 2d texture (x and y size)
83      glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR); // scale linearly
            when image bigger than texture
84      glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR); // scale linearly
            when image smalled than texture
85      glTexImage2D(GL_TEXTURE_2D, 0, 3, image[x]->sizeX, image[x]->sizeY, 0, GL_RGB,
            GL_UNSIGNED_BYTE, image[x]->data);
86      // 2d texture, 3 colors, width, height, RGB in that order, byte data, and the data.
87      gluBuild2DMipmaps(GL_TEXTURE_2D, 3, image[x]->sizeX, image[x]->sizeY, GL_RGB,
            GL_UNSIGNED_BYTE, image[x]->data);
88      x++;
89    }
90  }
91
92  /*
93   *  Load an image from .BMP file
94   *  reads image header to see if image is valid
95   *  has to be 24 bit bmp
96   */
97  int ImageLoad(char *filename, Image *image){
98      FILE *file;
99      unsigned long size;                 // size of the image in bytes.
100     unsigned long i;                    // standard counter.
101     unsigned short int planes;          // number of planes in image (must be 1)
102     unsigned short int bpp;             // number of bits per pixel (must be 24)
103     char temp;                          // used to convert bgr to rgb color.
104
105
106     // Make sure the file exists
```

```
107     if ((file = fopen(filename, "rb"))==NULL){
108  printf("File Not Found : %s\n",filename);
109  return 0;
110     }
111
112     if (file){
113  //if file exist get header
114  struct INFOHEADER bmpheader;
115  int width,height;
116  //printf("\nYeah..! Image opened\n");
117  fread(&bmpheader,sizeof(bmpheader),1,file);
118  //printf("\nImage width=%d\n",bmpheader.width);
119  //printf("\nImage Height=%d\n",bmpheader.height);
120  width = bmpheader.width;
121  height = bmpheader.height;
122  planes = bmpheader.planes;
123  bpp = bmpheader.bits;
124
125  // get the size (assuming 24 bpp)
126  size = bmpheader.imagesize;
127  //printf("width * height equals %lu \n", image->sizeX * image->sizeY * 3);
128  //printf("size of image equals %lu \n", size);
129
130  //set size in image class
131  image->sizeX = width;
132  image->sizeY = height;
133
134  //check if image is valid
135  if (planes != 1) {
136   printf("Planes from %s is not 1: %u\n", filename, planes);
137   return 0;
138  }
139
140  if (bpp != 24) {
141   printf("Bpp from %s is not 24: %u\n", filename, bpp);
142   //return 0;
143  }
144
145
146  /* move to start of image data */
147  rewind(file);
148  fseek(file, (long)bmpheader.offset_to_start_of_image_data_in_bytes, SEEK_SET);
149
150  // Allocate memory for the data
151  image->data = (char *) malloc(size);
152  if (image->data == NULL) {
153   printf("Error allocating memory for colour-corrected image data");
154   return 0;
155  }
156
157  // Read the data and set in allocated memory space
158  if ((i = fread(image->data,size,1,file)) != 1) {
159   printf("Error reading image data from %s.\n", filename);
160   return 0;
161  }
162
163  // reverse all of the colours bgr => rgb)
164  for (i=0;i<size;i+=3) {
165   temp = image->data[i];
166   image->data[i] = image->data[i+2];
167   image->data[i+2] = temp;
168  }
169     }
170     // Thats all folks
171     return 1;
172
173 }
```

A.5 interact.c

load the textures into the program

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>       // Header for sleeping

#include "interact.h"
#include "libfreenect.h"

int freenect_angle = 10;
int freenect_led;

void keyPressed(unsigned char key, int x, int y){
  /* avoid thrashing this procedure */
    usleep(100);
    //printf("key = %d\n", key);
  switch (key) {
    case 'Q':
    case 'q':
    case ESCAPE: // kill everything.


  //kill the kinect input
  //thread join hangs program. thats why it is commented out
  //pthread_join(freenect_thread, NULL);
  if (program == GLOBES){
    printf("escaping program = Holokin\n");
    //kill the windows
    if (DEBUG){
      glutDestroyWindow(window3);
    }
    glutDestroyWindow(window);
    glutDestroyWindow(window2);

    //release kinect
    free(depth_mid);
    free(depth_front);
    free(rgb_back);
    free(rgb_mid);
    free(rgb_front);
    pthread_exit(NULL);
  }

  if (program == KUBUS){
    printf("escaping program = kubus\n");
    glutDestroyWindow(window);
  }

  if (program == KINECTTEST){
    printf("escaping program = test\n");
    glutDestroyWindow(window3);
     //release kinect
    free(depth_mid);
    free(depth_front);
    free(rgb_back);
    free(rgb_mid);
    free(rgb_front);
    pthread_exit(NULL);
  }
  exit(1);
       break; // redundant.

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

    case KEY_DEL:
  if (type != AUTOSWITCH) type = AUTOSWITCH;
  else type = DEFAULTVIEW;
       break;

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

    case 'c':
    case 'C':
```

```
78  if (type != MANUALSWITCH) type = MANUALSWITCH;
79  else type = DEFAULTVIEW;
80        break;
81     case '+':
82  planet++;
83  if (planet == 11) planet = 0;
84        break;
85     case '-':
86  planet--;
87  if (planet == -1) planet = 10;
88        break;
89     case KEY_PAD_UP: //numpad up 8
90  xspeed += 0.01f;
91        break;
92     case KEY_PAD_DOWN: //numpad down 2
93  xspeed -= 0.01f;
94        break;
95     case KEY_PAD_LEFT: //numpad left 4
96  yspeed -= 0.01f;
97        break;
98     case KEY_PAD_RIGHT: //numpad right 6
99  yspeed += 0.01f;
100       break;
101    case KEY_PAD_MIDDLE: //numpad centre 5
102 yspeed = 0.0f;
103 xspeed = 0.0f;
104 zspeed = 0.0f;
105 zspeedmirror = 0.0f;
106 type = DEFAULTVIEW;
107       break;
108    case KEY_PAD_LEFT_UP: //numpad 7
109 zspeed -=0.01f;
110       break;
111    case KEY_PAD_RIGHT_UP: //numpad 9
112  zspeed +=0.01f;
113       break;
114    case KEY_PAD_LEFT_DOWN: //numpad 9
115  zspeedmirror -=0.01f;
116       break;
117    case KEY_PAD_RIGHT_DOWN: //numpad 9
118  zspeedmirror +=0.01f;
119       break;
120    case 'w':
121 freenect_angle++;
122 if (freenect_angle > 30) {
123   freenect_angle = 30;
124 }
125 freenect_set_tilt_degs(f_dev,freenect_angle);
126       break;
127    case 'x':
128 freenect_angle = 0;
129 freenect_set_tilt_degs(f_dev,freenect_angle);
130       break;
131    case 'f':
132 if (requested_format == FREENECT_VIDEO_IR_8BIT)
133   requested_format = FREENECT_VIDEO_RGB;
134 else if (requested_format == FREENECT_VIDEO_RGB)
135   requested_format = FREENECT_VIDEO_YUV_RGB;
136 else
137   requested_format = FREENECT_VIDEO_IR_8BIT;
138 freenect_set_tilt_degs(f_dev,freenect_angle);
139       break;
140    case 's':
141        freenect_angle--;
142 if (freenect_angle < -30) {
143   freenect_angle = -30;
144 }
145 freenect_set_tilt_degs(f_dev,freenect_angle);
146       break;
147    case ']':
148 if (cycle == 0) freenect_set_led(f_dev,LED_OFF);
149 if (cycle == 1) freenect_set_led(f_dev,LED_GREEN);
150 if (cycle == 2) freenect_set_led(f_dev,LED_RED);
151 if (cycle == 3) freenect_set_led(f_dev,LED_YELLOW);
152 if (cycle == 4) freenect_set_led(f_dev,LED_BLINK_GREEN);
153 if (cycle == 5) freenect_set_led(f_dev,LED_BLINK_GREEN);
154 if (cycle == 6) freenect_set_led(f_dev,LED_BLINK_RED_YELLOW);
```

```
155   if (cycle == 1) freenect_set_led(f_dev,LED_GREEN);
156   cycle++;
157   if (cycle == 7)cycle = 0;
158         break;
159       case 'm':
160       case 'M':
161           //moon = !moon;
162         break;
163       default:
164           printf ("Key %d pressed. No action there yet.\n", key);
165         break;
166       }
167 }
168
169 /* The function called whenever a normal key is pressed. */
170 void specialKeyPressed(int key, int x, int y){
171     /* avoid thrashing this procedure */
172     usleep(100);
173
174     switch (key) {
175     case GLUT_KEY_PAGE_UP: // move the cube into the distance.
176   z1-=0.02f;
177   break;
178
179     case GLUT_KEY_PAGE_DOWN: // move the cube closer.
180   z1+=0.02f;
181   break;
182
183     case GLUT_KEY_UP: // decrease x rotation speed;
184     z2-=0.02f;
185   break;
186
187     case GLUT_KEY_DOWN: // increase x rotation speed;
188     z2+=0.02f;
189   break;
190
191     case GLUT_KEY_LEFT: // decrease y or z rotation speed;
192     xmoon+=0.01f;
193   break;
194
195     case GLUT_KEY_RIGHT: // increase y of z rotation speed;
196     xmoon-=0.01f;
197   break;
198
199     case GLUT_KEY_END:
200     yspeed = 0.0f;
201     xspeed = 0.0f;
202     zspeed = 0.0f;
203     zspeedmirror = 0.0f;
204     yrot = 0;
205     xrot = 0;
206     zrot = 0;
207     zrotmirror = 0;
208     xmoon = 0.0f;
209     ymoon = 0.0f;
210     type = DEFAULTVIEW;
211     animate = GL_FALSE;
212   break;
213
214     case GLUT_KEY_HOME:
215   if (type != KEEPSPINNING){
216     type = KEEPSPINNING;
217           animate =!animate;
218   }
219   else {
220     type = DEFAULTVIEW;
221     animate =!animate;
222   }
223   break;
224
225      default:
226   break;
227     }
228 }
229
230 void MouseEvent(int button, int state, int x, int y){
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     }
243     glutPostRedisplay();
244 }
245
246 void MouseMotion(int x, int y){
247     static double acc;
248
249     if(moving){
250         if( key_modifiers & GLUT_ACTIVE_CTRL ) acc=0.02; else acc=1.0;
251         if(!animate){
252             yrot -=  (double)(y-start_y)*acc;
253             xrot +=  (double)(x-start_x)*acc;
254         } else {
255             yspeed = (double)(x-start_x)*0.05f;
256             xspeed = (double)(y-start_y)*0.05f;
257         }
258         start_x = x;
259         start_y = y;
260     }
261     glutPostRedisplay();
262 }
```

A.6 kinect.c

Kinect code, to register input

```
1
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 #include "libfreenect.h"
6 #include "kinect.h"
7
8 void kinect_set_active_area(int xlow, int xhigh, int ylow, int yhigh, int zlow, int
      zhigh)
9 {
10   xlowerbound = xlow;
11   xupperbound = xhigh;
12   ylowerbound = ylow;
13   yupperbound = yhigh;
14   zlowerbound = zlow;
15   zupperbound = zhigh;
16 }
17
18 void kinect_get_position()
19 {
20   int i, j;
21   for (i = 0; i < 480; i++){
22     for (j = 0; j < 640; j++){
23       //first define if in target area
24       if (location[i][j] < zupperbound || location[i][j] > zlowerbound){
25  loc[i][j] = 1;
26       }
27       else{
28  loc[i][j] = 0;
29       }
30     }
31   }
32   getBlobs();
33 }
34
35 void kinect_get_state()
```

```
36 {
37
38 }
39
40 void getMovement(){
41   static double acc;
42   acc = 0.3;
43   int reset;
44   //printf("moving: %i", moving);
45   if (!moving){
46     if (meanx == meanxoud && meany == meanyoud && meanx !=0 && meany !=0){
47       if (!animate && teller3 > 20){
48  teller3 = 0;
49  moving = 1;
50       }
51       else if (teller3 > 10){
52         teller3 = 0;
53  moving = 1;
54       }
55       teller3++;
56       //printf("teller3: %i", teller3);
57     }
58   }
59   if (moving){
60     if (meanx == 0 && meany == 0){
61       if (teller3 > 40){
62  teller3 = 0;
63  moving = 0;
64       }
65       else
66  teller3++;
67     }
68     else if(!animate){
69       xrot -=  (double)(meany-meanyoud*2)*acc+2;
70       yrot +=  (double)(meanx-meanxoud*2)*acc+2;
71     }
72     if (animate){
73       if (meanxoud != 0 && meanx !=0){
74  if (meanxoud - meanx < 0){
75    xspeed += 0.2f;
76    reset = 0;
77  }
78  else if (meanxoud - meanx > 0){
79    xspeed -= 0.2f;
80    reset = 0;
81  }
82       }
83       if(meanyoud != 0 && meany != 0){
84    if (meanyoud - meany > 0){
85      yspeed -= 0.2f;
86      reset = 0;
87    }
88    else if (meanyoud - meany < 0){
89      yspeed += 0.2f;
90      reset = 0;
91    }
92       }
93       //if (meanxoud == meanx && meanyoud == meany){
94  //if (reset == 5){
95    //yspeed -= 0;
96    //xspeed -= 0;
97    //reset = 0;
98  //}
99  //else reset++;
100       //}
101     }
102     if ((meanx == 0 || meany == 0) && (meanyoud == 0 || meanxoud == 0) && !animate){
103       //stop rotation
104       xspeed = 0;
105       yspeed = 0;
106       moving = 0;
107     }
108   }
109   meanxoud = meanx;
110   meanyoud = meany;
111 }
112
```

```
113 void emptyarray()
114 {
115     int i, j;
116     //empty arrays
117   for (i = 0; i < 240; i++){
118     for (j = 0; j < 320; j++){
119       smallblocks[i][j] = 0;
120     }
121   }
122   for (i = 0; i < 120; i++){
123     for (j = 0; j < 160; j++){
124       mediumblocks[i][j] = 0;
125     }
126   }
127   for (i = 0; i < 60; i++){
128     for (j = 0; j < 80; j++){
129       largeblocks[i][j] = 0;
130     }
131   }
132   for (i = 0; i < 30; i++){
133     for (j = 0; j < 40; j++){
134       bigblocks[i][j] = 0;
135     }
136   }
137   for (i = 0; i < 15; i++){
138     for (j = 0; j < 20; j++){
139       hugeblocks[i][j] = 0;
140     }
141   }
142   for (i = 0; i < 15; i++){
143     for (j = 0; j < 15; j++){
144       blob[i][j] = 0;
145     }
146   }
147 }
148
149
150 void getBlobs(){
151   printf("frame++\n");
152   int i, j;
153   int temp = 0;
154   int count= 0;
155
156
157   for (i = 0; i < 480; i++){
158     for (j = 0; j < 640; j++){
159       if (loc[i][j] != 1) count++;
160     }
161   }
162   printf("count= %i\n", count);
163   count = 0;
164
165   // first make square blocks
166   for (i = 0; i < 240; i++){
167     for (j = 0; j < 320; j++){
168       if (loc[2*i][2*j] != 1){
169 temp++;
170       }
171       if (loc[2*i+1][2*j] != 1){
172 temp++;
173       }
174       if (loc[2*i][2*j+1] != 1){
175 temp++;
176       }
177       if (loc[2*i+1][2*j+1] != 1){
178 temp++;
179       }
180       if (temp > 1){
181 smallblocks[i][j] = 1;
182 temp = 0;
183       }
184       else {
185 smallblocks[i][j] = 0;
186 temp = 0;
187       }
188     }
189   }
```

```
190   for (i = 0; i < 240; i++){
191     for (j = 0; j < 320; j++){
192       if (smallblocks[i][j] == 1) count++;
193     }
194   }
195   printf("smallcount= %i\n", count);
196   count = 0;
197
198   //make the blocks bigger
199   temp = 0;
200   for (i = 0; i < 120; i++){
201     for (j = 0; j < 160; j++){
202       if (smallblocks[2*i][2*j] == 1){
203 temp++;
204       }
205       if (smallblocks[2*i+1][2*j] == 1){
206 temp++;
207       }
208       if (smallblocks[2*i][2*j+1] == 1){
209 temp++;
210       }
211       if (smallblocks[2*i+1][2*j+1] == 1){
212 temp++;
213       }
214       if (temp > 1){
215 mediumblocks[i][j] = 1;
216 temp = 0;
217       }
218       else {
219 mediumblocks[i][j] = 0;
220 temp = 0;
221       }
222     }
223   }
224   for (i = 0; i < 120; i++){
225     for (j = 0; j < 160; j++){
226       if (mediumblocks[i][j] == 1) count++;
227     }
228   }
229   printf("mediumcount= %i\n", count);
230   count = 0;
231
232   //make the blocks bigger
233   temp = 0;
234   for (i = 0; i < 60; i++){
235     for (j = 0; j < 80; j++){
236       if (mediumblocks[2*i][2*j] == 1){
237 temp++;
238       }
239       if (mediumblocks[2*i+1][2*j] == 1){
240 temp++;
241       }
242       if (mediumblocks[2*i][2*j+1] == 1){
243 temp++;
244       }
245       if (mediumblocks[2*i+1][2*j+1] == 1){
246 temp++;
247       }
248       if (temp > 1){
249 largeblocks[i][j] = 1;
250 temp = 0;
251       }
252       else{
253 largeblocks[i][j] = 0;
254 temp = 0;
255       }
256     }
257   }
258   for (i = 0; i < 60; i++){
259     for (j = 0; j < 80; j++){
260       if (largeblocks[i][j] == 1) count++;
261     }
262   }
263   printf("largecount= %i\n", count);
264   count = 0;
265
266   //make the blocks bigger
```

```
267   temp = 0;
268   for (i = 0; i < 30; i++){
269     for (j = 0; j < 40; j++){
270       if (largeblocks[2*i][2*j] == 1){
271 temp++;
272       }
273       if (largeblocks[2*i+1][2*j] == 1){
274 temp++;
275       }
276       if (largeblocks[2*i][2*j+1] == 1){
277 temp++;
278       }
279       if (largeblocks[2*i+1][2*j+1] == 1){
280 temp++;
281       }
282       if (temp > 2){
283 bigblocks[i][j] = 1;
284 temp = 0;
285       }
286       else{
287 bigblocks[i][j] = 0;
288 temp = 0;
289       }
290     }
291   }
292   for (i = 0; i < 30; i++){
293     for (j = 0; j < 40; j++){
294       if (bigblocks[i][j] == 1) count++;
295     }
296   }
297   printf("bigcount= %i\n", count);
298   count = 0;
299
300   // big blocks
301   temp = 0;
302   for (i = 0; i < 15; i++){
303     for (j = 0; j < 20; j++){
304       if (bigblocks[2*i][2*j] == 1){
305 temp++;
306       }
307       if (bigblocks[2*i+1][2*j] == 1){
308 temp++;
309       }
310       if (bigblocks[2*i][2*j+1] == 1){
311 temp++;
312       }
313       if (bigblocks[2*i+1][2*j+1] == 1){
314 temp++;
315       }
316       if (temp > 1){
317 hugeblocks[i][j] = 1;
318 temp = 0;
319       }
320       else{
321 hugeblocks[i][j] = 0;
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= %i\n", count);
332   count = 0;
333
334   //now we throw away the left most 2 bits and the rightmost 3 bits
335   for(i = 0; i < 15; i++){
336     for (j = 0; j < 15; j++){
337       blob[i][j] = hugeblocks[i+2][j];
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
349    for (i = 0;i < 15; i++){
350      for (j = 0; j < 16; j ++){
351        if (hugeblocks[i][j] == 1){
352  row[i] = 1;
353  column[j] = 1;
354        }
355      }
356    }
357
358    printf("row: ");
359    for (i = 0; i < 15; i++){
360      printf("%i ,",row[i]);
361    }
362    printf("\n column: ");
363    for (j = 0; j < 15; j++){
364      printf("%i ,",column[j]);
365    }
366    printf("\n");
367
368    count = 0;
369    meanx = 0;
370    meany = 0;
371    //now we have values we can track
372    //calcuate mean of x
373    for (i = 0; i < 15; i++){
374      if (row[i] == 1){
375        count++;
376        meanx = meanx + i;
377      }
378    }
379    if(count != 0){
380      meanx = meanx/count;
381    }
382    count = 0;
383    for (j = 0; j < 15; j++){
384      if (column[j] == 1){
385        count++;
386        meany = meany + j;
387      }
388    }
389    if (count != 0){
390      meany = meany/count;
391    }
392
393    printf("meanx = %i meany = %i\n",meanx, meany);
394    printf("meanxoud = %i meanyoud = %i\n",meanxoud, meanyoud);
395    getMovement();
396  }
```

A.7 kubus2.c

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

```
1
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  #include "libfreenect.h"
6  #include "kinect.h"
7
8  void kinect_set_active_area(int xlow, int xhigh, int ylow, int yhigh, int zlow, int
       zhigh)
9  {
10   xlowerbound = xlow;
11   xupperbound = xhigh;
12   ylowerbound = ylow;
13   yupperbound = yhigh;
```

```
14    zlowerbound = zlow;
15    zupperbound = zhigh;
16  }
17
18  void kinect_get_position()
19  {
20    int i, j;
21    for (i = 0; i < 480; i++){
22      for (j = 0; j < 640; j++){
23        //first define if in target area
24        if (location[i][j] < zupperbound || location[i][j] > zlowerbound){
25   loc[i][j] = 1;
26        }
27        else{
28   loc[i][j] = 0;
29        }
30      }
31    }
32    getBlobs();
33  }
34
35  void kinect_get_state()
36  {
37
38  }
39
40  void getMovement(){
41    static double acc;
42    acc = 0.3;
43    int reset;
44    //printf("moving: %i", moving);
45    if (!moving){
46      if (meanx == meanxoud && meany == meanyoud && meanx !=0 && meany !=0){
47        if (!animate && teller3 > 20){
48   teller3 = 0;
49   moving = 1;
50        }
51        else if (teller3 > 10){
52          teller3 = 0;
53   moving = 1;
54        }
55        teller3++;
56        //printf("teller3: %i", teller3);
57      }
58    }
59    if (moving){
60      if (meanx == 0 && meany == 0){
61        if (teller3 > 40){
62   teller3 = 0;
63   moving = 0;
64        }
65        else
66   teller3++;
67      }
68      else if(!animate){
69        xrot -=  (double)(meany-meanyoud*2)*acc+2;
70        yrot +=  (double)(meanx-meanxoud*2)*acc+2;
71      }
72      if (animate){
73        if (meanxoud != 0 && meanx !=0){
74   if (meanxoud - meanx < 0){
75     xspeed += 0.2f;
76     reset = 0;
77   }
78   else if (meanxoud - meanx > 0){
79     xspeed -= 0.2f;
80     reset = 0;
81   }
82        }
83        if(meanyoud != 0 && meany != 0){
84     if (meanyoud - meany > 0){
85       yspeed -= 0.2f;
86       reset = 0;
87     }
88     else if (meanyoud - meany < 0){
89       yspeed += 0.2f;
90       reset = 0;
```

```
91      }
92          }
93          //if (meanxoud == meanx && meanyoud == meany){
94   //if (reset == 5){
95      //yspeed -= 0;
96      //xspeed -= 0;
97      //reset = 0;
98   //}
99   //else reset++;
100         //}
101     }
102     if ((meanx == 0 || meany == 0) && (meanyoud == 0 || meanxoud == 0) && !animate){
103       //stop rotation
104       xspeed = 0;
105       yspeed = 0;
106       moving = 0;
107     }
108   }
109   meanxoud = meanx;
110   meanyoud = meany;
111 }
112
113 void emptyarray()
114 {
115     int i, j;
116     //empty arrays
117   for (i = 0; i < 240; i++){
118     for (j = 0; j < 320; j++){
119       smallblocks[i][j] = 0;
120     }
121   }
122   for (i = 0; i < 120; i++){
123     for (j = 0; j < 160; j++){
124       mediumblocks[i][j] = 0;
125     }
126   }
127   for (i = 0; i < 60; i++){
128     for (j = 0; j < 80; j++){
129       largeblocks[i][j] = 0;
130     }
131   }
132   for (i = 0; i < 30; i++){
133     for (j = 0; j < 40; j++){
134       bigblocks[i][j] = 0;
135     }
136   }
137   for (i = 0; i < 15; i++){
138     for (j = 0; j < 20; j++){
139       hugeblocks[i][j] = 0;
140     }
141   }
142   for (i = 0; i < 15; i++){
143     for (j = 0; j < 15; j++){
144       blob[i][j] = 0;
145     }
146   }
147 }
148
149
150 void getBlobs(){
151   printf("frame++\n");
152   int i, j;
153   int temp = 0;
154   int count= 0;
155
156
157   for (i = 0; i < 480; i++){
158     for (j = 0; j < 640; j++){
159       if (loc[i][j] != 1) count++;
160     }
161   }
162   printf("count= %i\n", count);
163   count = 0;
164
165   // first make square blocks
166   for (i = 0; i < 240; i++){
167     for (j = 0; j < 320; j++){
```

```
168         if (loc[2*i][2*j] != 1){
169   temp++;
170         }
171         if (loc[2*i+1][2*j] != 1){
172   temp++;
173         }
174         if (loc[2*i][2*j+1] != 1){
175   temp++;
176         }
177         if (loc[2*i+1][2*j+1] != 1){
178   temp++;
179         }
180         if (temp > 1){
181   smallblocks[i][j] = 1;
182   temp = 0;
183         }
184         else {
185   smallblocks[i][j] = 0;
186   temp = 0;
187         }
188       }
189    }
190    for (i = 0; i < 240; i++){
191       for (j = 0; j < 320; j++){
192          if (smallblocks[i][j] == 1) count++;
193       }
194    }
195    printf("smallcount= %i\n", count);
196    count = 0;
197
198    //make the blocks bigger
199    temp = 0;
200    for (i = 0; i < 120; i++){
201       for (j = 0; j < 160; j++){
202          if (smallblocks[2*i][2*j] == 1){
203   temp++;
204         }
205         if (smallblocks[2*i+1][2*j] == 1){
206   temp++;
207         }
208         if (smallblocks[2*i][2*j+1] == 1){
209   temp++;
210         }
211         if (smallblocks[2*i+1][2*j+1] == 1){
212   temp++;
213         }
214         if (temp > 1){
215   mediumblocks[i][j] = 1;
216   temp = 0;
217         }
218         else {
219   mediumblocks[i][j] = 0;
220   temp = 0;
221         }
222       }
223    }
224    for (i = 0; i < 120; i++){
225       for (j = 0; j < 160; j++){
226          if (mediumblocks[i][j] == 1) count++;
227       }
228    }
229    printf("mediumcount= %i\n", count);
230    count = 0;
231
232    //make the blocks bigger
233    temp = 0;
234    for (i = 0; i < 60; i++){
235       for (j = 0; j < 80; j++){
236          if (mediumblocks[2*i][2*j] == 1){
237   temp++;
238         }
239         if (mediumblocks[2*i+1][2*j] == 1){
240   temp++;
241         }
242         if (mediumblocks[2*i][2*j+1] == 1){
243   temp++;
244         }
```

```
245         if (mediumblocks[2*i+1][2*j+1] == 1){
246   temp++;
247         }
248         if (temp > 1){
249   largeblocks[i][j] = 1;
250   temp = 0;
251         }
252         else{
253   largeblocks[i][j] = 0;
254   temp = 0;
255         }
256       }
257    }
258    for (i = 0; i < 60; i++){
259       for (j = 0; j < 80; j++){
260         if (largeblocks[i][j] == 1) count++;
261       }
262    }
263    printf("largecount= %i\n", count);
264    count = 0;
265
266    //make the blocks bigger
267    temp = 0;
268    for (i = 0; i < 30; i++){
269       for (j = 0; j < 40; j++){
270         if (largeblocks[2*i][2*j] == 1){
271   temp++;
272         }
273         if (largeblocks[2*i+1][2*j] == 1){
274   temp++;
275         }
276         if (largeblocks[2*i][2*j+1] == 1){
277   temp++;
278         }
279         if (largeblocks[2*i+1][2*j+1] == 1){
280   temp++;
281         }
282         if (temp > 2){
283   bigblocks[i][j] = 1;
284   temp = 0;
285         }
286         else{
287   bigblocks[i][j] = 0;
288   temp = 0;
289         }
290       }
291    }
292    for (i = 0; i < 30; i++){
293       for (j = 0; j < 40; j++){
294         if (bigblocks[i][j] == 1) count++;
295       }
296    }
297    printf("bigcount= %i\n", count);
298    count = 0;
299
300    // big blocks
301    temp = 0;
302    for (i = 0; i < 15; i++){
303       for (j = 0; j < 20; j++){
304         if (bigblocks[2*i][2*j] == 1){
305   temp++;
306         }
307         if (bigblocks[2*i+1][2*j] == 1){
308   temp++;
309         }
310         if (bigblocks[2*i][2*j+1] == 1){
311   temp++;
312         }
313         if (bigblocks[2*i+1][2*j+1] == 1){
314   temp++;
315         }
316         if (temp > 1){
317   hugeblocks[i][j] = 1;
318   temp = 0;
319         }
320         else{
321   hugeblocks[i][j] = 0;
```

```
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= %i\n", count);
332    count = 0;
333
334    //now we throw away the left most 2 bits and the rightmost 3 bits
335    for(i = 0; i < 15; i++){
336      for (j = 0; j < 15; j++){
337        blob[i][j] = hugeblocks[i+2][j];
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
349    for (i = 0;i < 15; i++){
350      for (j = 0; j < 16; j ++){
351        if (hugeblocks[i][j] == 1){
352  row[i] = 1;
353  column[j] = 1;
354        }
355      }
356    }
357
358    printf("row: ");
359    for (i = 0; i < 15; i++){
360      printf("%i ,",row[i]);
361    }
362    printf("\n column: ");
363    for (j = 0; j < 15; j++){
364      printf("%i ,",column[j]);
365    }
366    printf("\n");
367
368    count = 0;
369    meanx = 0;
370    meany = 0;
371    //now we have values we can track
372    //calcuate mean of x
373    for (i = 0; i < 15; i++){
374      if (row[i] == 1){
375        count++;
376        meanx = meanx + i;
377      }
378    }
379    if(count != 0){
380      meanx = meanx/count;
381    }
382    count = 0;
383    for (j = 0; j < 15; j++){
384      if (column[j] == 1){
385        count++;
386        meany = meany + j;
387      }
388    }
389    if (count != 0){
390      meany = meany/count;
391    }
392
393    printf("meanx = %i meany = %i\n",meanx, meany);
394    printf("meanxoud = %i meanyoud = %i\n",meanxoud, meanyoud);
395    getMovement();
396  }
```

A.8 test.c

Program to check if the kinect input works

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <math.h>
4  #include <GL/glut.h>
5  #include <GL/gl.h>
6  #include <GL/glu.h>
7
8  #include "libfreenect.h"
9  #include "threadfunc.h"
10 #include "drawX.h"
11
12
13
14 void *gl_threadfunc2(void *arg)
15 {
16   glutInit(&g_argc, g_argv);
17   glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |  GLUT_DEPTH );
18
19   glutInitWindowSize(1280, 480);
20   glutInitWindowPosition(0, 0);
21   window3 = glutCreateWindow("LibFreenect");
22   glutDisplayFunc(&DrawGLScene);
23   glutIdleFunc(&DrawGLScene);
24   glutReshapeFunc(&ReSizeGLScene);
25   /* Register the function called when the keyboard is pressed. */
26   glutKeyboardFunc(&keyPressed);
27   /* Register the function called when special keys (arrows, page down, etc) are pressed
         . */
28   glutSpecialFunc(&specialKeyPressed);
29   InitGL(1280, 480);
30
31   glutMainLoop();
32   return NULL;
33 }
34
35 int main(int argc, char **argv){
36   int res;
37   program = KINECTTEST;
38   depth_mid = (uint8_t*)malloc(640*480*3);
39   depth_front = (uint8_t*)malloc(640*480*3);
40   rgb_back = (uint8_t*)malloc(640*480*3);
41   rgb_mid = (uint8_t*)malloc(640*480*3);
42   rgb_front = (uint8_t*)malloc(640*480*3);
43
44   printf("Kinect camera test\n");
45
46   int i;
47   for (i=0; i<2048; i++) {
48    float v = i/2048.0;
49    v = powf(v, 3)* 6;
50    t_gamma[i] = v*6*256;
51   }
52
53   g_argc = argc;
54   g_argv = argv;
55
56   if (freenect_init(&f_ctx, NULL) < 0) {
57    printf("freenect_init() failed\n");
58    return 1;
59   }
60
61   freenect_set_log_level(f_ctx, FREENECT_LOG_DEBUG);
62
63   int nr_devices = freenect_num_devices (f_ctx);
64   printf ("Number of devices found: %d\n", nr_devices);
65
66   int user_device_number = 0;
67   if (argc > 1)
68    user_device_number = atoi(argv[1]);
69
70   if (nr_devices < 1)
71    return 1;
```

```
72
73    if (freenect_open_device(f_ctx, &f_dev, user_device_number) < 0) {
74      printf("Could not open device\n");
75      return 1;
76    }
77
78    res = pthread_create(&freenect_thread, NULL, freenect_threadfunc, NULL);
79    if (res) {
80      printf("pthread_create failed\n");
81      return 1;
82    }
83
84    // OS X requires GLUT to run on the main thread
85    gl_threadfunc2(NULL);
86
87    return 0;
88
89 }
```

A.9 defines.h

Code file with all defines

```
1
2 #ifndef DEFINES_H
3 #define DEFINES_H
4
5 #define FREENECT_COUNTS_PER_G 819 /**< Ticks per G for accelerometer as set per http://
      www.kionix.com/Product%20Sheets/KXSD9%20Product%20Brief.pdf */
6 #define TRUE  1
7 #define FALSE 0
8 #define ESCAPE 27
9 #define PAGE_UP 73
10 #define PAGE_DOWN 81
11 #define UP_ARROW 72
12 #define DOWN_ARROW 80
13 #define LEFT_ARROW 75
14 #define RIGHT_ARROW 77
15 #define KEY_DEL 127
16 #define KEY_PAD_UP 56
17 #define KEY_PAD_DOWN 50
18 #define KEY_PAD_LEFT 52
19 #define KEY_PAD_RIGHT 54
20 #define KEY_PAD_MIDDLE 53
21 #define KEY_PAD_LEFT_UP 55
22 #define KEY_PAD_LEFT_DOWN 49
23 #define KEY_PAD_RIGHT_UP 57
24 #define KEY_PAD_RIGHT_DOWN 51
25 #define MAX_IMAGES 48
26
27 #define KEEPSPINNING 0
28 #define AUTOSWITCH 1
29 #define MANUALSWITCH 2
30 #define DEFAULTVIEW 4
31
32 #define KUBUS 0
33 #define GLOBES 1
34 #define KINECTTEST 2
35
36
37 // change DEBUG to 1 if you want to see the kinect feed in Glview
38 #define DEBUG 0
39
40 #endif
```

A.10 drawX.h

Code of drawX.h

```
1
2 #ifndef DRAW_H
3 #define DRAW_H
4 #include "libfreenect.h"
5
6 int window, window2, window3;  // The numbers of our GLUT windows
7 extern GLfloat  ymoon;
8 extern GLfloat  xmoon;
9 extern GLfloat  ymoonrot;
10 extern GLfloat  xmoonrot;
11 extern GLfloat  yrot;    // X Rotation
12 extern GLfloat  xrot;    // Y Rotation
13 extern GLfloat  zrot;    // Z Rotation
14 extern GLfloat  zrotmirror;    // Z Rotation
15 extern GLfloat  xspeed;    // x rotation speed
16 extern GLfloat  yspeed;    // y rotation speed
17 extern GLfloat  zspeed;    // Z rotation speed
18 extern GLfloat  zspeedmirror;    // Z rotation speed
19 extern GLfloat  z1;    // depth into the screen.
20 extern GLfloat  z2;    // depth into the screen.
21 extern GLshort transparent;
22 extern GLshort moon;
23
24 extern int type;   // 0 for animate, 1 for switching textures, 2 for manually change
25 extern int teller;    // counter to control the pace of changing textures.
26 extern int teller2;    // counter to control the pace of changing textures.
27 extern int last;   // last shown was the earth
28 extern int last2;    // last shown was the earth
29 extern int teller3;
30 extern int planet;    // for manually changing the planets a counter to keep track of
       which planet we look at.
31 extern int bmpx;    // startnr of texture in second window
32 extern int bmpx2;    // startnr of texture in first window
33 extern int cycle;    // variable to keep track of wich led color the kinect must show
34
35
36 GLUquadricObj  *quadratic;    // Storage For Our Quadratic Objects
37 GLUquadricObj  *quadratic2;    // Storage For Our Quadratic Objects
38 GLUquadricObj  *quadraticmoon;    // Storage For Our Quadratic Objects
39
40
41 //funtion prototypes
42 void DrawGLScene();
43 void DrawGLScene2();
44 void DrawGLScene3();
45 void DrawGLScene4();
46 void InitGL2(GLsizei Width, GLsizei Height);
47 void InitGL(int Width, int Height);
48 void ReSizeGLScene(GLsizei Width, GLsizei Height);
49 void ReSizeGLScene2(GLsizei Width, GLsizei Height);
50
51
52 #endif
```

A.11 imageload.h

Code of imageload.h

```
1
2 #ifndef IMAGELOAD_H
3 #define IMAGELOAD_H
4
5 #include "libfreenect.h"
6
7 char *filenamearray[MAX_IMAGES]; // array with imagelocations
8 GLuint texture[MAX_IMAGES];  // Storage for texture.
9
10 /* Image type - contains height, width and data */
11 struct Image {
12  unsigned long sizeX;
13  unsigned long sizeY;
14  char *data;
```

```
15 };
16 typedef struct Image Image;
17
18 /*
19  * Struct infoheader
20  * found at
21  *  http://iamfeelinlucky.blogspot.nl/2009/09/c-code-to-read-bmp-header-information.html
22  */
23 struct INFOHEADER
24 {
25       unsigned short int signature;
26       unsigned int size;
27       unsigned short int reserved1;
28       unsigned short int reserved2;
29       int offset_to_start_of_image_data_in_bytes;
30       int size_of_BITMAPINFOHEADER;
31       int width,height;
32       unsigned short int planes;
33       unsigned short int bits;
34       unsigned int compression;
35       unsigned int imagesize;
36       int xresolution,yresolution;
37       unsigned int ncolours;
38       unsigned int importantcolours;
39 }__attribute__((__packed__));
40 typedef struct INFOHEADER INFOHEADER;
41
42 int location[640][480];
43
44 Image *image[MAX_IMAGES]; // variable for the images
45
46 void FillArrayImageNames();
47 int ImageLoad(char *filename, Image *image);
48 void LoadGLTextures();
49
50
51 #endif
```

A.12 interact.h

Code of interact.h

```
1
2 #ifndef INTERACT_H
3 #define INTERACT_H
4 #include "libfreenect.h"
5
6 int light;     // Lighting ON/OFF
7 int update;
8 int meanxoud;
9 int meanyoud;
10 int meanx;
11 int meany;
12
13 extern GLboolean animate; // When true movement continues till it reguarly slowed down
14 int     moving, start_x, start_y; // variables to keep track of mousemovement
15 int      key_modifiers;   // variable to check if there is movement
16
17 extern int    type;   // 0 for animate, 1 for switching textures, 2 for manually change
18 extern int    teller;   // counter to control the pace of changing textures.
19 extern int    teller2;   // counter to control the pace of changing textures.
20 extern int    last;   // last shown was the earth
21 extern int    last2;   // last shown was the earth
22
23 extern int    planet;   // for manually changing the planets a counter to keep track of
       which planet we look at.
24 extern int    bmpx;   // startnr of texture in second window
25 extern int    bmpx2;   // startnr of texture in first window
26
27 void keyPressed(unsigned char key, int x, int y);
28 void specialKeyPressed(int key, int x, int y);
29 void MouseEvent(int button, int state, int x, int y);
30 void MouseMotion(int x, int y);
```

```
31
32
33 #endif
```

A.13 kinect.h

Code of kinect.h

```
1
2 #ifndef KINECT_H
3 #define KINECT_H
4
5   int xlowerbound;
6   int xupperbound;
7   int ylowerbound;
8   int yupperbound;
9   int zlowerbound;
10  int zupperbound;
11
12  int loc[480][640];
13  int smallblocks[240][320];
14  int mediumblocks[120][160];
15  int largeblocks[60][80];
16  int bigblocks[30][40];
17  int hugeblocks[15][20];
18  int blob[15][15];
19
20  void kinect_get_position();
21  void kinect_get_state();
22  void kinect_set_active_area(int xlow, int xhigh, int ylow, int yhigh, int zlow, int
        zhigh);
23  void getMovement();
24  void getBlobs();
25  void emptyarray();
26
27 #endif
```

A.14 libfreenect.h

Code of libfreenect.h

```
1 /*
2  * This file is part of the OpenKinect Project. http://www.openkinect.org
3  *
4  * Copyright (c) 2010 individual OpenKinect contributors. See the CONTRIB file
5  * for details.
6  *
7  * This code is licensed to you under the terms of the Apache License, version
8  * 2.0, or, at your option, the terms of the GNU General Public License,
9  * version 2.0. See the APACHE20 and GPL2 files for the text of the licenses,
10 * or the following URLs:
11 * http://www.apache.org/licenses/LICENSE-2.0
12 * http://www.gnu.org/licenses/gpl-2.0.txt
13 *
14 * If you redistribute this file in source form, modified or unmodified, you
15 * may:
16 *   1) Leave this header intact and distribute it under the same terms,
17 *      accompanying it with the APACHE20 and GPL20 files, or
18 *   2) Delete the Apache 2.0 clause and accompany it with the GPL2 file, or
19 *   3) Delete the GPL v2 clause and accompany it with the APACHE20 file
20 * In all cases you must keep the copyright notice intact and include a copy
21 * of the CONTRIB file.
22 *
23 * Binary distributions must follow the binary distribution requirements of
24 * either License.
25 */
26
27 #ifndef LIBFREENECT_H
```

```
28 #define LIBFREENECT_H
29
30 #include <stdint.h>
31 #include <GL/glut.h>
32 #include <GL/gl.h>
33 #include <GL/glu.h>
34 #include "defines.h"
35 #include "interact.h"
36 #include "drawX.h"
37 #include "imageload.h"
38 #include "kinect.h"
39 #include "threadfunc.h"
40
41 #ifdef __cplusplus
42 extern "C" {
43 #endif
44 extern int freenect_angle;
45 pthread_t freenect_thread;
46 extern volatile int die;
47 // back: owned by libfreenect (implicit for depth)
48 // mid: owned by callbacks, "latest frame ready"
49 // front: owned by GL, "currently being drawn"
50 uint8_t *depth_mid, *depth_front;
51 uint8_t *rgb_back, *rgb_mid, *rgb_front;
52
53 GLuint gl_depth_tex;
54 GLuint gl_rgb_tex;
55
56 uint16_t t_gamma[2048];
57 int g_argc;
58 char **g_argv;
59
60 int program;
61
62 /// Enumeration of available resolutions.
63 /// Not all available resolutions are actually supported for all video formats.
64 /// Frame modes may not perfectly match resolutions.  For instance,
65 /// FREENECT_RESOLUTION_MEDIUM is 640x488 for the IR camera.
66 typedef enum {
67  FREENECT_RESOLUTION_LOW    = 0, /**< QVGA - 320x240 */
68  FREENECT_RESOLUTION_MEDIUM = 1, /**< VGA  - 640x480 */
69  FREENECT_RESOLUTION_HIGH   = 2, /**< SXGA - 1280x1024 */
70  FREENECT_RESOLUTION_DUMMY  = 2147483647, /**< Dummy value to force enum to be 32 bits
        wide */
71 } freenect_resolution;
72
73 /// Enumeration of video frame information states.
74 /// See http://openkinect.org/wiki/Protocol_Documentation#RGB_Camera for more
        information.
75 typedef enum {
76  FREENECT_VIDEO_RGB              = 0, /**< Decompressed RGB mode (demosaicing done by
        libfreenect) */
77  FREENECT_VIDEO_BAYER            = 1, /**< Bayer compressed mode (raw information from
        camera) */
78  FREENECT_VIDEO_IR_8BIT          = 2, /**< 8-bit IR mode  */
79  FREENECT_VIDEO_IR_10BIT         = 3, /**< 10-bit IR mode */
80  FREENECT_VIDEO_IR_10BIT_PACKED = 4, /**< 10-bit packed IR mode */
81  FREENECT_VIDEO_YUV_RGB          = 5, /**< YUV RGB mode */
82  FREENECT_VIDEO_YUV_RAW          = 6, /**< YUV Raw mode */
83  FREENECT_VIDEO_DUMMY            = 2147483647, /**< Dummy value to force enum to be 32
        bits wide */
84 } freenect_video_format;
85
86 /// Enumeration of depth frame states
87 /// See http://openkinect.org/wiki/Protocol_Documentation#RGB_Camera for more
        information.
88 typedef enum {
89  FREENECT_DEPTH_11BIT          = 0, /**< 11 bit depth information in one uint16_t/pixel */
90  FREENECT_DEPTH_10BIT          = 1, /**< 10 bit depth information in one uint16_t/pixel */
91  FREENECT_DEPTH_11BIT_PACKED = 2, /**< 11 bit packed depth information */
92  FREENECT_DEPTH_10BIT_PACKED = 3, /**< 10 bit packed depth information */
93  FREENECT_DEPTH_DUMMY          = 2147483647, /**< Dummy value to force enum to be 32 bits
        wide */
94 } freenect_depth_format;
95
96 /// Structure to give information about the width, height, bitrate,
97 /// framerate, and buffer size of a frame in a particular mode, as
```

```
98 /// well as the total number of bytes needed to hold a single frame.
99 typedef struct {
100  uint32_t reserved;                /**< unique ID used internally.  The meaning of values
       may change without notice.  Don't touch or depend on the contents of this field.
       We mean it. */
101  freenect_resolution resolution; /**< Resolution this freenect_frame_mode describes,
       should you want to find it again with freenect_find_*_frame_mode(). */
102  union {
103   int32_t dummy;
104   freenect_video_format video_format;
105   freenect_depth_format depth_format;
106  };                                /**< The video or depth format that this
       freenect_frame_mode describes.  The caller should know which of video_format or
       depth_format to use, since they called freenect_get_*_frame_mode() */
107  int32_t bytes;                    /**< Total buffer size in bytes to hold a single frame
       of data.  Should be equivalent to width * height * (data_bits_per_pixel+
       padding_bits_per_pixel) / 8 */
108  int16_t width;                    /**< Width of the frame, in pixels */
109  int16_t height;                   /**< Height of the frame, in pixels */
110  int8_t data_bits_per_pixel;       /**< Number of bits of information needed for each
       pixel */
111  int8_t padding_bits_per_pixel;  /**< Number of bits of padding for alignment used for
       each pixel */
112  int8_t framerate;                 /**< Approximate expected frame rate, in Hz */
113  int8_t is_valid;                  /**< If 0, this freenect_frame_mode is invalid and does
        not describe a supported mode.  Otherwise, the frame_mode is valid. */
114 } freenect_frame_mode;
115
116 /// Enumeration of LED states
117 /// See http://openkinect.org/wiki/Protocol_Documentation#Setting_LED for more
       information.
118 typedef enum {
119  LED_OFF              = 0, /**< Turn LED off */
120  LED_GREEN            = 1, /**< Turn LED to Green */
121  LED_RED              = 2, /**< Turn LED to Red */
122  LED_YELLOW           = 3, /**< Turn LED to Yellow */
123  LED_BLINK_GREEN      = 4, /**< Make LED blink Green */
124  LED_BLINK_RED      = 5, /**< Make LED blink Red */
125  LED_BLINK_RED_YELLOW = 6, /**< Make LED blink Red/Yellow */
126 } freenect_led_options;
127
128
129 /// Enumeration of tilt motor status
130 typedef enum {
131  TILT_STATUS_STOPPED = 0x00, /**< Tilt motor is stopped */
132  TILT_STATUS_LIMIT   = 0x01, /**< Tilt motor has reached movement limit */
133  TILT_STATUS_MOVING  = 0x04, /**< Tilt motor is currently moving to new position */
134 } freenect_tilt_status_code;
135
136 /// Data from the tilt motor and accelerometer
137 typedef struct {
138  int16_t                    accelerometer_x; /**< Raw accelerometer data for X-axis, see
       FREENECT_COUNTS_PER_G for conversion */
139  int16_t                    accelerometer_y; /**< Raw accelerometer data for Y-axis, see
       FREENECT_COUNTS_PER_G for conversion */
140  int16_t                    accelerometer_z; /**< Raw accelerometer data for Z-axis, see
       FREENECT_COUNTS_PER_G for conversion */
141  int8_t                     tilt_angle;      /**< Raw tilt motor angle encoder
       information */
142  freenect_tilt_status_code tilt_status;      /**< State of the tilt motor (stopped,
       moving, etc...) */
143 } freenect_raw_tilt_state;
144
145 struct _freenect_context;
146 typedef struct _freenect_context freenect_context; /**< Holds information about the usb
       context. */
147
148 struct _freenect_device;
149 typedef struct _freenect_device freenect_device; /**< Holds device information. */
150
151 // usb backend specific section
152 #ifdef _WIN32
153   /* frees Windows users of the burden of specifying the path to <libusb-1.0/libusb.h>
        */
154   typedef void freenect_usb_context;
155 #else
156   #include <libusb-1.0/libusb.h>
```

```
157   typedef libusb_context freenect_usb_context; /**< Holds libusb-1.0 specific
          information */
158 #endif
159 //
160
161 /// If Win32, export all functions for DLL usage
162 #ifndef _WIN32
163   #define FREENECTAPI /**< DLLExport information for windows, set to nothing on other
          platforms */
164 #else
165   /**< DLLExport information for windows, set to nothing on other platforms */
166   #ifdef __cplusplus
167     #define FREENECTAPI extern "C" __declspec(dllexport)
168   #else
169     // this is required when building from a Win32 port of gcc without being
170     // forced to compile all of the library files (.c) with g++...
171     #define FREENECTAPI __declspec(dllexport)
172   #endif
173 #endif
174
175 /// Enumeration of message logging levels
176 typedef enum {
177  FREENECT_LOG_FATAL = 0,      /**< Log for crashing/non-recoverable errors */
178  FREENECT_LOG_ERROR,          /**< Log for major errors */
179  FREENECT_LOG_WARNING,        /**< Log for warning messages */
180  FREENECT_LOG_NOTICE,         /**< Log for important messages */
181  FREENECT_LOG_INFO,           /**< Log for normal messages */
182  FREENECT_LOG_DEBUG,          /**< Log for useful development messages */
183  FREENECT_LOG_SPEW,           /**< Log for slightly less useful messages */
184  FREENECT_LOG_FLOOD,          /**< Log EVERYTHING. May slow performance. */
185 } freenect_loglevel;
186
187
188
189 /**
190  * Initialize a freenect context and do any setup required for
191  * platform specific USB libraries.
192  *
193  * @param ctx Address of pointer to freenect context struct to allocate and initialize
194  * @param usb_ctx USB context to initialize. Can be NULL if not using multiple contexts.
195  *
196  * @return 0 on success, < 0 on error
197  */
198 FREENECTAPI int freenect_init(freenect_context **ctx, freenect_usb_context *usb_ctx);
199
200 /**
201  * Closes the device if it is open, and frees the context
202  *
203  * @param ctx freenect context to close/free
204  *
205  * @return 0 on success
206  */
207 FREENECTAPI int freenect_shutdown(freenect_context *ctx);
208
209 /// Typedef for logging callback functions
210 typedef void (*freenect_log_cb)(freenect_context *dev, freenect_loglevel level, const
        char *msg);
211
212 /**
213  * Set the log level for the specified freenect context
214  *
215  * @param ctx context to set log level for
216  * @param level log level to use (see freenect_loglevel enum)
217  */
218 FREENECTAPI void freenect_set_log_level(freenect_context *ctx, freenect_loglevel level);
219
220 /**
221  * Callback for log messages (i.e. for rerouting to a file instead of
222  * stdout)
223  *
224  * @param ctx context to set log callback for
225  * @param cb callback function pointer
226  */
227 FREENECTAPI void freenect_set_log_callback(freenect_context *ctx, freenect_log_cb cb);
228
229 /**
230  * Calls the platform specific usb event processor
```

```
231  *
232  * @param ctx context to process events for
233  *
234  * @return 0 on success, other values on error, platform/library dependant
235  */
236 FREENECTAPI int freenect_process_events(freenect_context *ctx);
237
238 /**
239  * Return the number of kinect devices currently connected to the
240  * system
241  *
242  * @param ctx Context to access device count through
243  *
244  * @return Number of devices connected, < 0 on error
245  */
246 FREENECTAPI int freenect_num_devices(freenect_context *ctx);
247
248 /**
249  * Opens a kinect device via a context. Index specifies the index of
250  * the device on the current state of the bus. Bus resets may cause
251  * indexes to shift.
252  *
253  * @param ctx Context to open device through
254  * @param dev Device structure to assign opened device to
255  * @param index Index of the device on the bus
256  *
257  * @return 0 on success, < 0 on error
258  */
259 FREENECTAPI int freenect_open_device(freenect_context *ctx, freenect_device **dev, int
        index);
260
261 /**
262  * Closes a device that is currently open
263  *
264  * @param dev Device to close
265  *
266  * @return 0 on success
267  */
268 FREENECTAPI int freenect_close_device(freenect_device *dev);
269
270 /**
271  * Set the device user data, for passing generic information into
272  * callbacks
273  *
274  * @param dev Device to attach user data to
275  * @param user User data to attach
276  */
277 FREENECTAPI void freenect_set_user(freenect_device *dev, void *user);
278
279 /**
280  * Retrieve the pointer to user data from the device struct
281  *
282  * @param dev Device from which to get user data
283  *
284  * @return Pointer to user data
285  */
286 FREENECTAPI void *freenect_get_user(freenect_device *dev);
287
288 /// Typedef for depth image received event callbacks
289 typedef void (*freenect_depth_cb)(freenect_device *dev, void *depth, uint32_t timestamp)
        ;
290 /// Typedef for video image received event callbacks
291 typedef void (*freenect_video_cb)(freenect_device *dev, void *video, uint32_t timestamp)
        ;
292
293 /**
294  * Set callback for depth information received event
295  *
296  * @param dev Device to set callback for
297  * @param cb Function pointer for processing depth information
298  */
299 FREENECTAPI void freenect_set_depth_callback(freenect_device *dev, freenect_depth_cb cb)
        ;
300
301 /**
302  * Set callback for video information received event
303  *
```

```
304  * @param dev Device to set callback for
305  * @param cb Function pointer for processing video information
306  */
307 FREENECTAPI void freenect_set_video_callback(freenect_device *dev, freenect_video_cb cb)
        ;
308
309 /**
310  * Set the buffer to store depth information to. Size of buffer is
311  * dependant on depth format. See FREENECT_DEPTH_*_SIZE defines for
312  * more information.
313  *
314  * @param dev Device to set depth buffer for.
315  * @param buf Buffer to store depth information to.
316  *
317  * @return 0 on success, < 0 on error
318  */
319 FREENECTAPI int freenect_set_depth_buffer(freenect_device *dev, void *buf);
320
321 /**
322  * Set the buffer to store depth information to. Size of buffer is
323  * dependant on video format. See FREENECT_VIDEO_*_SIZE defines for
324  * more information.
325  *
326  * @param dev Device to set video buffer for.
327  * @param buf Buffer to store video information to.
328  *
329  * @return 0 on success, < 0 on error
330  */
331 FREENECTAPI int freenect_set_video_buffer(freenect_device *dev, void *buf);
332
333 /**
334  * Start the depth information stream for a device.
335  *
336  * @param dev Device to start depth information stream for.
337  *
338  * @return 0 on success, < 0 on error
339  */
340 FREENECTAPI int freenect_start_depth(freenect_device *dev);
341
342 /**
343  * Start the video information stream for a device.
344  *
345  * @param dev Device to start video information stream for.
346  *
347  * @return 0 on success, < 0 on error
348  */
349 FREENECTAPI int freenect_start_video(freenect_device *dev);
350
351 /**
352  * Stop the depth information stream for a device
353  *
354  * @param dev Device to stop depth information stream on.
355  *
356  * @return 0 on success, < 0 on error
357  */
358 FREENECTAPI int freenect_stop_depth(freenect_device *dev);
359
360 /**
361  * Stop the video information stream for a device
362  *
363  * @param dev Device to stop video information stream on.
364  *
365  * @return 0 on success, < 0 on error
366  */
367 FREENECTAPI int freenect_stop_video(freenect_device *dev);
368
369 /**
370  * Updates the accelerometer state using a blocking control message
371  * call.
372  *
373  * @param dev Device to get accelerometer data from
374  *
375  * @return 0 on success, < 0 on error. Accelerometer data stored to
376  * device struct.
377  */
378 FREENECTAPI int freenect_update_tilt_state(freenect_device *dev);
379
```

```
380 /**
381  * Retrieve the tilt state from a device
382  *
383  * @param dev Device to retrieve tilt state from
384  *
385  * @return The tilt state struct of the device
386  */
387 FREENECTAPI freenect_raw_tilt_state* freenect_get_tilt_state(freenect_device *dev);
388
389 /**
390  * Return the tilt state, in degrees with respect to the horizon
391  *
392  * @param state The tilt state struct from a device
393  *
394  * @return Current degree of tilt of the device
395  */
396 FREENECTAPI double freenect_get_tilt_degs(freenect_raw_tilt_state *state);
397
398 /**
399  * Set the tilt state of the device, in degrees with respect to the
400  * horizon. Uses blocking control message call to update
401  * device. Function return does not reflect state of device, device
402  * may still be moving to new position after the function returns. Use
403  * freenect_get_tilt_status() to find current movement state.
404  *
405  * @param dev Device to set tilt state
406  * @param angle Angle the device should tilt to
407  *
408  * @return 0 on success, < 0 on error.
409  */
410 FREENECTAPI int freenect_set_tilt_degs(freenect_device *dev, double angle);
411
412 /**
413  * Return the movement state of the tilt motor (moving, stopped, etc...)
414  *
415  * @param state Raw state struct to get the tilt status code from
416  *
417  * @return Status code of the tilt device. See
418  * freenect_tilt_status_code enum for more info.
419  */
420 FREENECTAPI freenect_tilt_status_code freenect_get_tilt_status(freenect_raw_tilt_state *
        state);
421
422 /**
423  * Set the state of the LED. Uses blocking control message call to
424  * update device.
425  *
426  * @param dev Device to set the LED state
427  * @param option LED state to set on device. See freenect_led_options enum.
428  *
429  * @return 0 on success, < 0 on error
430  */
431 FREENECTAPI int freenect_set_led(freenect_device *dev, freenect_led_options option);
432
433 /**
434  * Get the axis-based gravity adjusted accelerometer state, as laid
435  * out via the accelerometer data sheet, which is available at
436  *
437  * http://www.kionix.com/Product%20Sheets/KXSD9%20Product%20Brief.pdf
438  *
439  * @param state State to extract accelerometer data from
440  * @param x Stores X-axis accelerometer state
441  * @param y Stores Y-axis accelerometer state
442  * @param z Stores Z-axis accelerometer state
443  */
444 FREENECTAPI void freenect_get_mks_accel(freenect_raw_tilt_state *state, double* x,
        double* y, double* z);
445
446 /**
447  * Get the number of video camera modes supported by the driver.  This includes both RGB
        and IR modes.
448  *
449  * @return Number of video modes supported by the driver
450  */
451 FREENECTAPI int freenect_get_video_mode_count();
452
453 /**
```

```
454  * Get the frame descriptor of the nth supported video mode for the
455  * video camera.
456  *
457  * @param n Which of the supported modes to return information about
458  *
459  * @return A freenect_frame_mode describing the nth video mode
460  */
461 FREENECTAPI const freenect_frame_mode freenect_get_video_mode(int mode_num);
462
463 /**
464  * Get the frame descriptor of the current video mode for the specified
465  * freenect device.
466  *
467  * @param dev Which device to return the currently-set video mode for
468  *
469  * @return A freenect_frame_mode describing the current video mode of the specified
         device
470  */
471 FREENECTAPI const freenect_frame_mode freenect_get_current_video_mode(freenect_device *
       dev);
472
473 /**
474  * Convenience function to return a mode descriptor matching the
475  * specified resolution and video camera pixel format, if one exists.
476  *
477  * @param res Resolution desired
478  * @param fmt Pixel format desired
479  *
480  * @return A freenect_frame_mode that matches the arguments specified, if such a valid
         mode exists; otherwise, an invalid freenect_frame_mode.
481  */
482 FREENECTAPI const freenect_frame_mode freenect_find_video_mode(freenect_resolution res,
       freenect_video_format fmt);
483
484 /**
485  * Sets the current video mode for the specified device.  If the
486  * freenect_frame_mode specified is not one provided by the driver
487  * e.g. from freenect_get_video_mode() or freenect_find_video_mode()
488  * then behavior is undefined.  The current video mode cannot be
489  * changed while streaming is active.
490  *
491  * @param dev Device for which to set the video mode
492  * @param mode Frame mode to set
493  *
494  * @return 0 on success, < 0 if error
495  */
496 FREENECTAPI int freenect_set_video_mode(freenect_device* dev, const freenect_frame_mode
       mode);
497
498 /**
499  * Get the number of depth camera modes supported by the driver.  This includes both RGB
         and IR modes.
500  *
501  * @return Number of depth modes supported by the driver
502  */
503 FREENECTAPI int freenect_get_depth_mode_count();
504
505 /**
506  * Get the frame descriptor of the nth supported depth mode for the
507  * depth camera.
508  *
509  * @param n Which of the supported modes to return information about
510  *
511  * @return A freenect_frame_mode describing the nth depth mode
512  */
513 FREENECTAPI const freenect_frame_mode freenect_get_depth_mode(int mode_num);
514
515 /**
516  * Get the frame descriptor of the current depth mode for the specified
517  * freenect device.
518  *
519  * @param dev Which device to return the currently-set depth mode for
520  *
521  * @return A freenect_frame_mode describing the current depth mode of the specified
         device
522  */
```

```
523 FREENECTAPI const freenect_frame_mode freenect_get_current_depth_mode(freenect_device *
      dev);
524
525 /**
526  * Convenience function to return a mode descriptor matching the
527  * specified resolution and depth camera pixel format, if one exists.
528  *
529  * @param res Resolution desired
530  * @param fmt Pixel format desired
531  *
532  * @return A freenect_frame_mode that matches the arguments specified, if such a valid
       mode exists; otherwise, an invalid freenect_frame_mode.
533  */
534 FREENECTAPI const freenect_frame_mode freenect_find_depth_mode(freenect_resolution res,
      freenect_depth_format fmt);
535
536 /**
537  * Sets the current depth mode for the specified device.  The mode
538  * cannot be changed while streaming is active.
539  *
540  * @param dev Device for which to set the depth mode
541  * @param mode Frame mode to set
542  *
543  * @return 0 on success, < 0 if error
544  */
545 FREENECTAPI int freenect_set_depth_mode(freenect_device* dev, const freenect_frame_mode
      mode);
546
547
548 freenect_context *f_ctx;
549 freenect_device *f_dev;
550 freenect_video_format requested_format;
551 freenect_video_format current_format;
552
553
554 #ifdef __cplusplus
555 }
556 #endif
557
558 #endif //
```

A.15 threadfunc.h

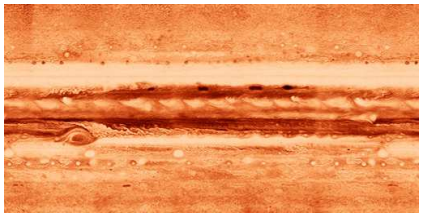
Code of threadfunc.h

```
1 #ifndef THREADFUNC_H
2 #define THREADFUNC_H
3 #include "libfreenect.h"
4
5 extern pthread_mutex_t gl_backbuf_mutex;
6 extern pthread_cond_t gl_frame_cond;
7 extern int got_rgb;
8 extern int got_depth;
9
10
11 //function prototypes
12 void *gl_threadfunc(void *arg);
13 //void depth_cb(freenect_device *dev, void *v_depth, uint32_t timestamp);
14 //rgb_cb(freenect_device *dev, void *rgb, uint32_t timestamp);
15 void *freenect_threadfunc(void *arg);
16
17 #endif
```

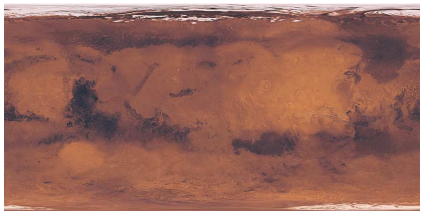
B Appendix B

B.1 textures

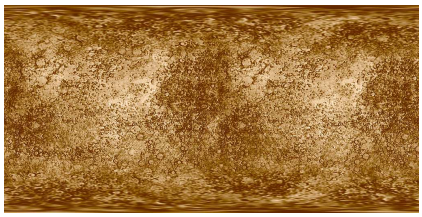
(a) Earth



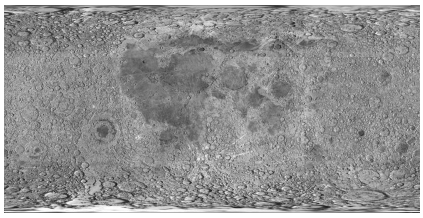
(b) Jupiter



(c) Mars



(d) Mercury



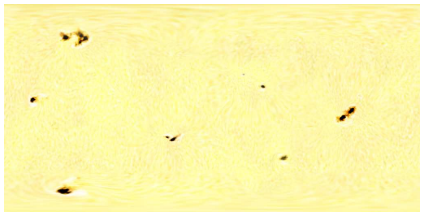
(e) Moon

(f) Neptune

(g) Pluto

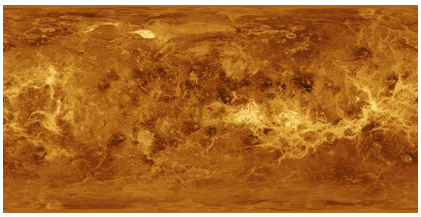
(h) Saturn

Figure 11: Textures



(a) Sun

(b) Uranus



(c) Venus

Figure 12: Textures