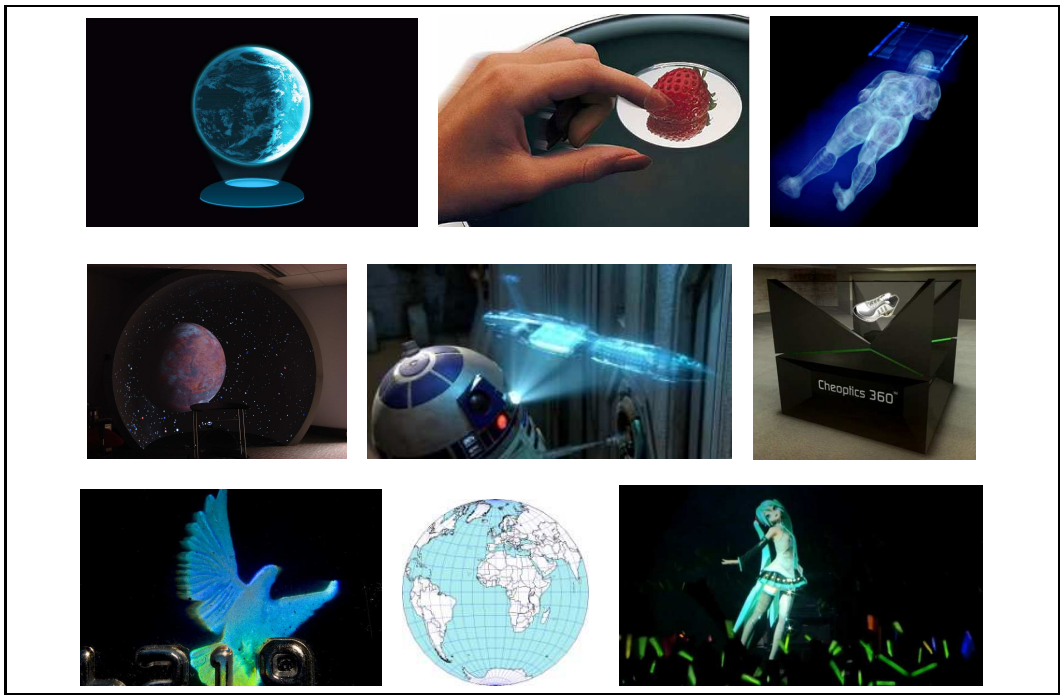
Holography and Kinect

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.

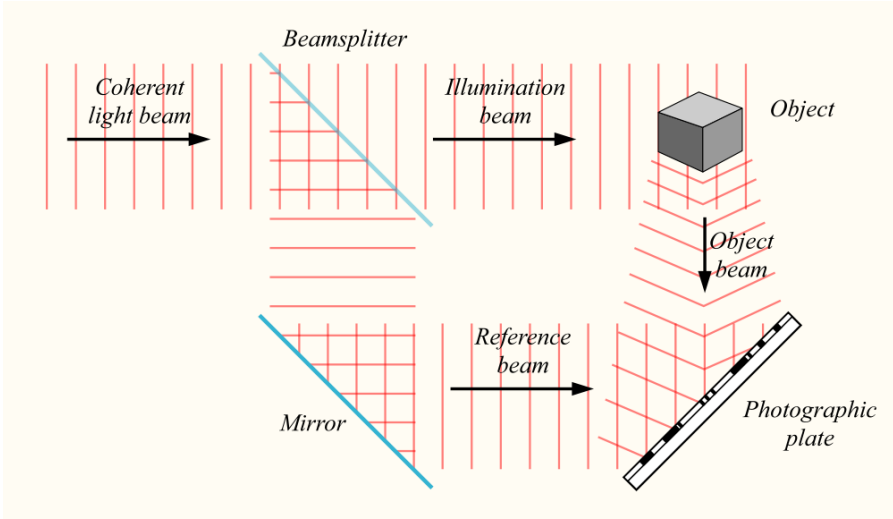
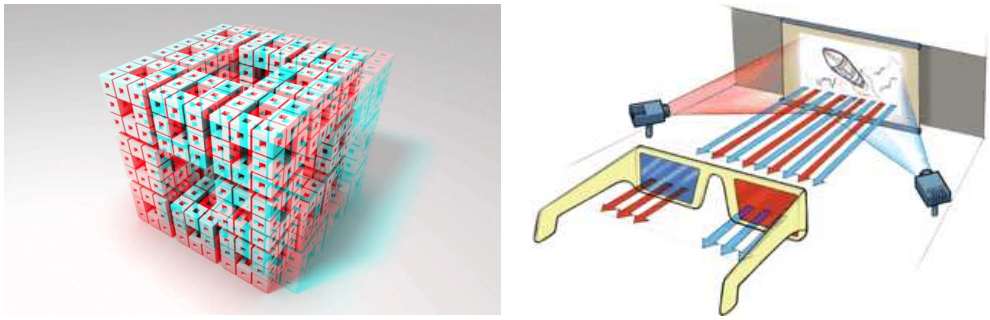


Figure 1: Recording the interference image

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].

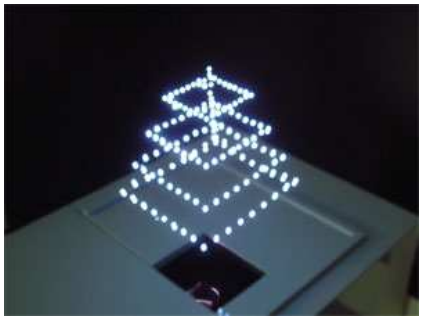
A glass pyramid with projections on all sides gives the illusion of an object inside the pyramid like the viZoo™ Cheoptics hologram [19] (see Figure 3a). An image is projected on top of the pyramid such that every side of the pyramid has a part of the object shown onto it. These images give the illusion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

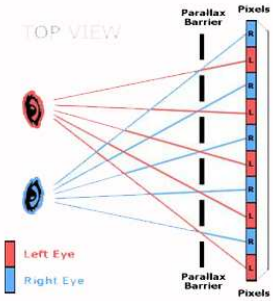
Figure 3: Cheoptics 360 from ViZoo and Star Wars hologram R2D2

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.

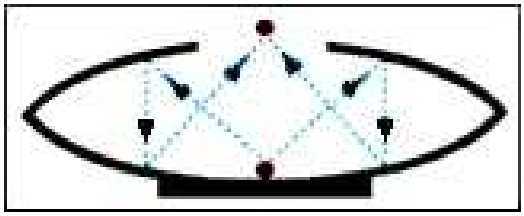


Figure 5: Concept optical illusion using a mirascope

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.

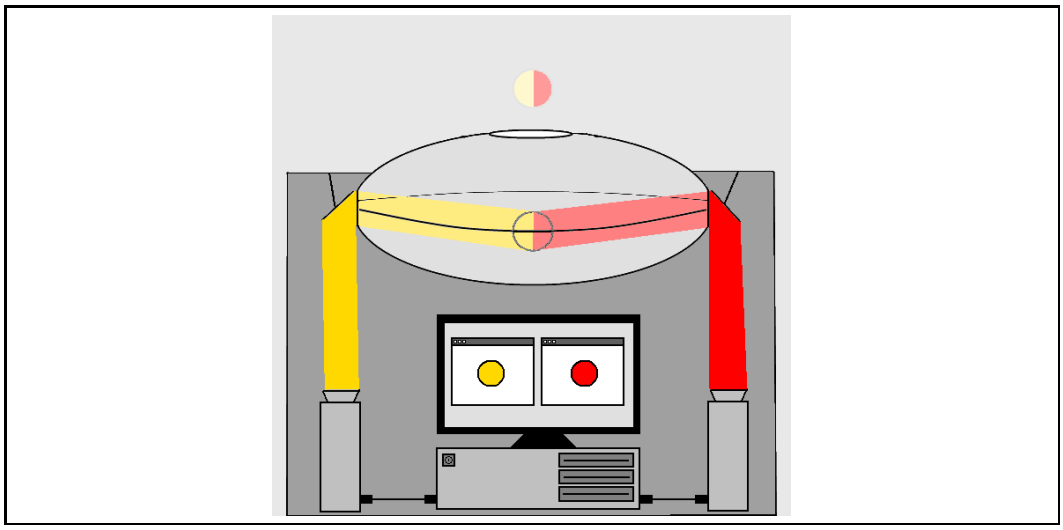


Figure 6: Concept optical illusion with beamer

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.

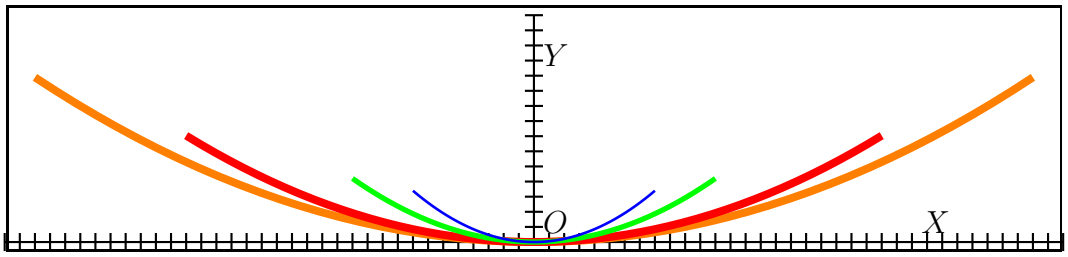


Figure 7: Graph showing different sizes of mirascopes

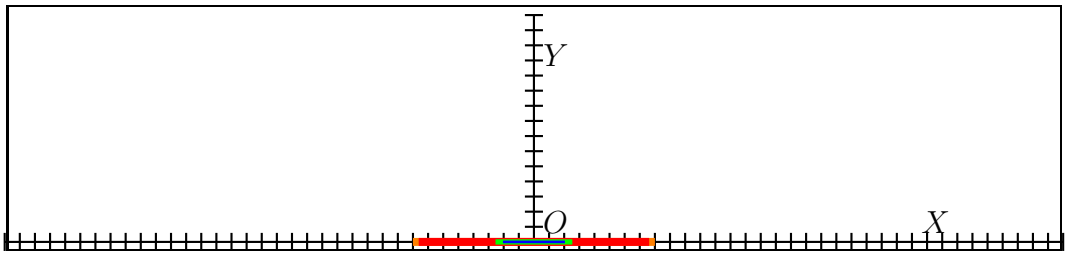


Figure 8: Graph showing apertures of different sizes of mirascopes

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.

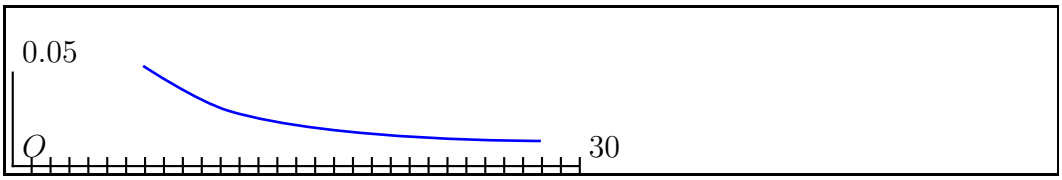


Figure 9: Graph showing relation between diameter/2 and factor a in $F(x) = ax^2 + bx + c$

Table 2: Aperture height relation

Total height	Aperture	Aperture/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.

```c
/*
 * This file is part of the OpenKinect Project. http://www.openkinect.org
 *
 * Copyright (c) 2010 individual OpenKinect contributors. See the CONTRIB file
 * for details.
 *
 * This code is licensed to you under the terms of the Apache License, version
 * 2.0, or, at your option, the terms of the GNU General Public License,
 * version 2.0. See the APACHE20 and GPL2 files for the text of the licenses,
 * or the following URLs:
 * http://www.apache.org/licenses/LICENSE-2.0
 * http://www.gnu.org/licenses/gpl-2.0.txt
 *
 * If you redistribute this file in source form, modified or unmodified, you
 * may:
 *   1) Leave this header intact and distribute it under the same terms,
 *      accompanying it with the APACHE20 and GPL20 files, or
 *   2) Delete the Apache 2.0 clause and accompany it with the GPL2 file, or
 *   3) Delete the GPL v2 clause and accompany it with the APACHE20 file
 * In all cases you must keep the copyright notice intact and include a copy
 * of the CONTRIB file.
 *
 * Binary distributions must follow the binary distribution requirements of
 * either License.
 */


#include <stdio.h>
#include <pthread.h>
#include <math.h>

#include "libfreenect.h"
#include "threadfunc.h"


int main(int argc, char **argv){
 program = GLOBES;
 FillArrayImageNames();
 //set frame in which to recognize gestures
 kinect_set_active_area(0,480,0,640,550,650);
 update = 0;
 meanxoud = 0;
 meanyoud = 0;
 int res;

 depth_mid = (uint8_t*)malloc(640*480*3);
 depth_front = (uint8_t*)malloc(640*480*3);
 rgb_back = (uint8_t*)malloc(640*480*3);
 rgb_mid = (uint8_t*)malloc(640*480*3);
 rgb_front = (uint8_t*)malloc(640*480*3);

 printf("Kinect camera test\n");

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

 g_argc = argc;
 g_argv = argv;

 if (freenect_init(&f_ctx, NULL) < 0) {
  printf("freenect_init() failed\n");
  return 1;
 }

```

```
  freenect_set_log_level(f_ctx, FREENECT_LOG_DEBUG);

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

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

  if (nr_devices < 1)
   return 1;

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

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

  // OS X requires GLUT to run on the main thread
  gl_threadfunc(NULL);

  return 0;
}
```

A.2 threadfunc.c

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

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

pthread_mutex_t gl_backbuf_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t gl_frame_cond = PTHREAD_COND_INITIALIZER;


freenect_video_format requested_format = FREENECT_VIDEO_RGB;
freenect_video_format current_format = FREENECT_VIDEO_RGB;

int got_rgb = 0;
int got_depth = 0;

void *gl_threadfunc(void *arg){
  glutInit(&g_argc, g_argv);
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |  GLUT_DEPTH );

  /* get a 640 x 480 window */
  glutInitWindowSize(640, 480);
  /* the window starts at the upper left corner of the screen */
  glutInitWindowPosition(0, 0);
  /* Open a window */
  window = glutCreateWindow("2");  // first create the reacting window.
  /* Register the function to do all our OpenGL drawing. */
  glutDisplayFunc(&DrawGLScene2);
  /* Even if there are no events, redraw our gl scene. */
  glutIdleFunc(&DrawGLScene2);
  /* Register the function called when our window is resized. */
  glutReshapeFunc(&ReSizeGLScene2);
  /* Register the function called when the keyboard is pressed. */
  glutKeyboardFunc(&keyPressed);
  /* Register the function called when mouse left is klicked */
  glutMouseFunc(&MouseEvent);
```

```
  /* Register the function called when special keys (arrows, page down, etc) are pressed. */
  glutSpecialFunc(&specialKeyPressed);
  InitGL2(640, 480);

  /* the window starts at the upper left corner of the screen */
  glutInitWindowPosition(640, 0);
  /* Open a second window */
  window2 = glutCreateWindow("1");
   /* Even if there are no events, redraw our gl scene. */
  glutDisplayFunc(&DrawGLScene4);
    /* Even if there are no events, redraw our gl scene. */
  glutIdleFunc(&DrawGLScene4);
  /* Register the function called when our window is resized. */
  glutReshapeFunc(&ReSizeGLScene2);
  /* Register the function called when the keyboard is pressed. */
  glutKeyboardFunc(&keyPressed);
  /* Register the function called when mouse left is klicked */
  glutMouseFunc(&MouseEvent);
  /* Register the function called when special keys (arrows, page down, etc) are pressed. */
  glutSpecialFunc(&specialKeyPressed);
  glutMotionFunc(MouseMotion);
  /* Initialize our window. */
  InitGL2(640, 480);

  if (DEBUG == 1){
    glutInitWindowSize(1280, 480);
    glutInitWindowPosition(0, 520);
    window3 = glutCreateWindow("LibFreenect");
    glutDisplayFunc(&DrawGLScene);
    glutIdleFunc(&DrawGLScene);
    glutReshapeFunc(&ReSizeGLScene);
    /* Register the function called when the keyboard is pressed. */
    glutKeyboardFunc(&keyPressed);
    /* Register the function called when mouse left is klicked */
    glutMouseFunc(&MouseEvent);
    /* Register the function called when special keys (arrows, page down, etc) are pressed. */
    glutSpecialFunc(&specialKeyPressed);
    InitGL(1280, 480);
  }
  glutMainLoop();
  return NULL;
}
//

void depth_cb(freenect_device *dev, void *v_depth, uint32_t timestamp)
{
 int i,j;
 uint16_t *depth = (uint16_t*)v_depth;
 //save the depth map in a two dimensional array for that is easy to work with
 for (i = 0; i < 480; i++){
   for (j = 0; j < 640; j++){
       location[i][j] = depth[i*640+j];
   }
 }

 pthread_mutex_lock(&gl_backbuf_mutex);
 for (i=0; i<640*480; i++) {
  int pval = t_gamma[depth[i]];
  int lb = pval & 0xff;
  switch (pval>>8) {
   case 0:
       depth_mid[3*i+0] = 255;
       depth_mid[3*i+1] = 255-lb;
       depth_mid[3*i+2] = 255-lb;
     break;
   case 1:
       depth_mid[3*i+0] = 255;
       depth_mid[3*i+1] = lb;
       depth_mid[3*i+2] = 0;
     break;
   case 2:
       depth_mid[3*i+0] = 255-lb;
       depth_mid[3*i+1] = 255;
       depth_mid[3*i+2] = 0;
```

```
      break;
    case 3:
        depth_mid[3*i+0] = 0;
        depth_mid[3*i+1] = 255;
        depth_mid[3*i+2] = lb;
      break;
    case 4:
        depth_mid[3*i+0] = 0;
        depth_mid[3*i+1] = 255-lb;
        depth_mid[3*i+2] = 255;
      break;
    case 5:
        depth_mid[3*i+0] = 0;
        depth_mid[3*i+1] = 0;
        depth_mid[3*i+2] = 255-lb;
      break;
    default:
      depth_mid[3*i+0] = 0;
      depth_mid[3*i+1] = 0;
      depth_mid[3*i+2] = 0;
      break;
   }
  }
  got_depth++;
  pthread_cond_signal(&gl_frame_cond);
  pthread_mutex_unlock(&gl_backbuf_mutex);
}

void rgb_cb(freenect_device *dev, void *rgb, uint32_t timestamp)
{
  pthread_mutex_lock(&gl_backbuf_mutex);
  // swap buffers
  assert (rgb_back == rgb);
  rgb_back = rgb_mid;
  freenect_set_video_buffer(dev, rgb_back);
  rgb_mid = (uint8_t*)rgb;

  got_rgb++;
  pthread_cond_signal(&gl_frame_cond);
  pthread_mutex_unlock(&gl_backbuf_mutex);
}

void *freenect_threadfunc(void *arg)
{
  int accelCount = 0;

  freenect_set_tilt_degs(f_dev,freenect_angle);
  freenect_set_led(f_dev,LED_RED);
  freenect_set_depth_callback(f_dev, depth_cb);
  freenect_set_video_callback(f_dev, rgb_cb);
  freenect_set_video_mode(f_dev, freenect_find_video_mode(FREENECT_RESOLUTION_MEDIUM, current_format));
  freenect_set_depth_mode(f_dev, freenect_find_depth_mode(FREENECT_RESOLUTION_MEDIUM, FREENECT_DEPTH_11BIT));
  freenect_set_video_buffer(f_dev, rgb_back);

  freenect_start_depth(f_dev);
  freenect_start_video(f_dev);

  //printf("'w'-tilt up, 's'-level, 'x'-tilt down, '0'-'6'-select LED mode, 'f'-video format\n");

  while (freenect_process_events(f_ctx) >= 0) {
   //Throttle the text output
   if (accelCount++ >= 2000)
   {
    accelCount = 0;
    freenect_raw_tilt_state* state;
    freenect_update_tilt_state(f_dev);
    state = freenect_get_tilt_state(f_dev);
    double dx,dy,dz;
    freenect_get_mks_accel(state, &dx, &dy, &dz);
    //printf("\r raw acceleration: %4d %4d %4d  mks acceleration: %4f %4f %4f", state->accelerometer_x, state->accelerometer_y, state->accelerometer_z, dx, dy, dz);
    fflush(stdout);
   }

```

```
  if (requested_format != current_format) {
   freenect_stop_video(f_dev);
   freenect_set_video_mode(f_dev, freenect_find_video_mode(FREENECT_RESOLUTION_MEDIUM,
        requested_format));
   freenect_start_video(f_dev);
   current_format = requested_format;
  }
 }

 //printf("\nshutting down streams...\n");

 freenect_stop_depth(f_dev);
 freenect_stop_video(f_dev);

 freenect_close_device(f_dev);
 freenect_shutdown(f_ctx);

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

A.3 drawX.c

This is the main draw code.

```
#include <stdio.h>
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <pthread.h>
#include <math.h>
#include "libfreenect.h"
#include "drawX.h"
#include "threadfunc.h"

int overlayproject = 20;
int interval = 0;

GLboolean animate = GL_FALSE;
int type = 2;   // 0 for animate, 1 for switching textures, 2 for manually change
int teller = 700;  // counter to control the pace of changing textures.
int teller2 = 700;  // counter to control the pace of changing textures.
int last = 2;   // last shown was the earth
int last2 = 12;   // last shown was the earth
int teller3 = 0;
int planet = 2;   // for manually changing the planets a counter to keep track of which
     planet we look at.
int bmpx = 10;   // startnr of texture in second window
int bmpx2 = 0;   // startnr of texture in first window
int cycle = 0;   // variable to keep track of wich led color the kinect must show

GLfloat xmoon = 0.0f;
GLfloat ymoon = 0.0f;
GLfloat xmoonrot = 0.0f;
GLfloat ymoonrot = 0.0f;
GLfloat yrot = 0.0f;  // X Rotation
GLfloat xrot = 0.0f;  // Y Rotation
GLfloat zrot = 0.0f;  // Z Rotation
GLfloat zrotmirror = 0.0f; // Z Rotation mirrored
GLfloat xspeed = 0.0f;  // x rotation speed
GLfloat yspeed = 0.0f;  // y rotation speed
GLfloat zspeed = 0.0f;  // Z rotation speed
GLfloat zspeedmirror = 0.0f; // Z rotation speed mirrored
GLfloat z1=-3.8f;  // depth into the screen.
GLfloat z2=-3.8f;  // depth into the screen.
GLshort transparent = FALSE;
GLshort moon = FALSE;
GLfloat twoscreen = 1.95f; //shift to left or to the right to accomodate 2 monitor
     version
GLfloat correctie = 0.0f; //variable to correct viewpoint while lateral turning
int framecounter = 0;

//Light declarations
```

```
/* white ambient light at half intensity (rgba) */
GLfloat LightAmbient[] = { 1.5f, 1.5f, 1.5f, 1.0f };
/* super bright, full intensity diffuse light. */
GLfloat LightDiffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };
/* position of light (x, y, z, (position of light)) */
GLfloat LightPosition[] = { 1.0f, 1.0f, 2.0f, 1.0f };

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

    yrot+=xspeed;                    // X Axis Rotation
    xrot+=yspeed;                    // Y Axis Rotation
    zrot+=zspeed;     // Z Axis Rotation
    zrotmirror+=zspeedmirror;

    if (yrot > 360.0f || yrot < -360.0f) yrot = 0.0f;
    if (xrot > 360.0f || xrot < -360.0f) xrot = 0.0f;
    if (zrot > 360.0f || zrot < -360.0f) zrot = 0.0f;
    if (zrotmirror > 360.0f || zrotmirror < -360.0f) zrotmirror = 0.0f;

    ymoonrot+=xmoon;
    xmoonrot+=ymoon;

    // slow down after a while.
    if (xspeed > 0) xspeed = xspeed - 0.0001f;
    else if (xspeed < 0) xspeed = xspeed + 0.0001f;
    if (yspeed > 0) yspeed = yspeed - 0.0001f;
    else if (yspeed < 0) yspeed = yspeed + 0.0001f;
    if (zspeed > 0) zspeed = zspeed - 0.0001f;
    else if (zspeed < 0) zspeed = zspeed + 0.0001f;
    if (zspeedmirror > 0) zspeedmirror = zspeedmirror - 0.0001f;
    else if (zspeedmirror < 0) zspeedmirror = zspeedmirror + 0.0001f;
    if (xmoon > 0) xmoonrot = xmoonrot - 0.0001f;
    else if (xmoon < 0) xmoon = xmoon + 0.0001f;
    if (ymoon > 0) ymoon = ymoon - 0.0001f;
    else if (ymoon < 0) ymoon = ymoon + 0.0001f;


    bmpx2 = last;
    //A switch to select what texture has to be shown on the solid sphere
    switch (type){
 case KEEPSPINNING: // animate
     glBindTexture(GL_TEXTURE_2D, texture[2]);
   break;
 case AUTOSWITCH: //switching textures
     if (teller2 == 0) teller2 = 700;
     if (teller2 % 50 == 0){
       teller2--;
       last++;
       if (last > 10) last = 0;
     }
     else teller2--;
     glBindTexture(GL_TEXTURE_2D, texture[bmpx2]);
   break;
 case MANUALSWITCH: //manually switch textures
   switch (planet) {
     case 0:  //mercury
  glBindTexture(GL_TEXTURE_2D, texture[0]);
  last = 0;
       break;
     case 1:  //venus
  glBindTexture(GL_TEXTURE_2D, texture[1]);
  last = 1;
       break;
     case 2:  //earth
  glBindTexture(GL_TEXTURE_2D, texture[2]);
  last = 2;
       break;
     case 3:  //mars
  glBindTexture(GL_TEXTURE_2D, texture[3]);
  last = 3;
       break;
     case 4:  //jupiter
  glBindTexture(GL_TEXTURE_2D, texture[4]);
```

```
  last = 4;
        break;
      case 5:  //saturn
  glBindTexture(GL_TEXTURE_2D, texture[5]);
  last = 5;
        break;
      case 6:  //uranus
  glBindTexture(GL_TEXTURE_2D, texture[6]);
  last = 6;
        break;
      case 7:  //neptune
  glBindTexture(GL_TEXTURE_2D, texture[7]);
  last = 7;
        break;
      case 8:  //pluto
  glBindTexture(GL_TEXTURE_2D, texture[8]);
  last = 8;
        break;
      case 9:  //sun
        glBindTexture(GL_TEXTURE_2D, texture[9]);
          last = 9;
        break;
      case 10:  //moon
        glBindTexture(GL_TEXTURE_2D, texture[46]);
          last = 46;
        break;
      default:
        glBindTexture(GL_TEXTURE_2D, texture[2]);
        break;
      }
   break;
 default:
      glBindTexture(GL_TEXTURE_2D, texture[last]);   // choose the texture to use.
   break;
    }
    //make sure globe fills the maximum amount of the screen.
    //glTranslatef(-1.8f,0.0f,z1);            // move z units into the screen.
    glTranslatef(0.0f,0.0f,z1);            // move z units into the screen.
    glTranslatef(twoscreen,0.0f,0.0f);  // move twoscreen units to the side
    GLfloat hulp = fmodf(yrot, 360.0f);

    if (hulp != 0){
      correctie = 1/hulp;
      if (correctie > 1.0f || correctie < -1.0f){
 correctie = 0.0f;
      }
      if (correctie > 0.042){
 correctie = 0.042;
      }
      if (correctie < -0.042){
 correctie = -0.042;
      }
    }

    if (DEBUG && framecounter == 15){
      printf("yrot = %f, xrot = %f, zrot = %f, correctie = %f\n", yrot, xrot, zrot,
          correctie);
      framecounter = 0;
    }
    framecounter++;
    //the rotation function regarding the speed
    glRotatef(yrot,-1.0f,0.0f,-0.42f + correctie);  // Rotate On The X Axis
    glRotatef(xrot,0.0f,1.0f,0.0f);     // Rotate On The Y Axis
    glRotatef(zrot,-1.0f,1.0f,-0.42f + correctie);  // Rotate on the Z Axis
    glRotatef(zrotmirror,1.0f,1.0f,0.42f + correctie); // Rotate on the Z mirror Axis

    //turn globe into 'normal position'
    glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
    glRotatef(-45.0f, 0.0f, 0.0f, 1.0f);
    gluSphere(quadratic,1.3f,32,32);                  // Draw A Sphere

    //check if overlayproject needs to be drawn
    if (transparent) {
      //enable alpha testing
      glEnable(GL_ALPHA_TEST);
      //enable blending
      glEnable (GL_BLEND);
```

```
      // set kind of blening
      glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
      //select texture for second bigger transparent sphere
      if (animate){
 if (overlayproject == 42) {
   overlayproject = 20;
 }
 glBindTexture(GL_TEXTURE_2D, texture[overlayproject]);   // choose the texture to use.
 if (interval > 10){
   overlayproject++;
   interval = 0;
 }
 interval++;
      }
      else {
 glBindTexture(GL_TEXTURE_2D, texture[overlayproject]);   // choose the texture to use
      }

      //draw a second sphere on top of the first sphere
      gluSphere(quadratic2,1.4f,32,32);
      //disable alpha testing
      glDisable(GL_ALPHA_TEST);
      //disable blening
      glDisable (GL_BLEND);
    }

    if (moon){
      //glRotatef(-yrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
      //glRotatef(-xrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
      //glRotatef(-zrot,0.0f,0.0f,1.0f);  // Rotate on the Z Axis

      glTranslatef(0.0f,3.0f,0.0f);            // move x units into the screen.
      glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
      //glRotatef(ymoonrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
      //glRotatef(xmoonrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
      glBindTexture(GL_TEXTURE_2D, texture[46]);

      gluSphere(quadraticmoon,0.2f,32,32);          // Draw A moon
      //glRotatef(-xmoonrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
      //glRotatef(-ymoonrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
      glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
      glTranslatef(0.0,-3.0f,0.0f);            // move x units into the screen.
    }
    //set lighting
    if (!light) {
      glDisable(GL_LIGHTING);
    } else {
      glEnable(GL_LIGHTING);
    }

    glutPostRedisplay();
    // since this is double buffered, swap the buffers to display what just got drawn.
    glutSwapBuffers();
}

void DrawGLScene4()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  // Clear The Screen And The
        Depth Buffer
    glLoadIdentity();     // Reset The View

    yrot+=xspeed;                    // X Axis Rotation
    xrot+=yspeed;                    // Y Axis Rotation
    zrot+=zspeed;     // Z Axis Rotation
    zrotmirror+= zspeedmirror;

    if (yrot > 360.0f || yrot < -360.0f) yrot = 0.0f;
    if (xrot > 360.0f || xrot < -360.0f) xrot = 0.0f;
    if (zrot > 360.0f || zrot < -360.0f) zrot = 0.0f;
    if (zrotmirror > 360.0f || zrotmirror < -360.0f) zrotmirror = 0.0f;

    ymoonrot+=xmoon;
    xmoonrot+=ymoon;

    // slow down after a while.
    if (xspeed > 0) xspeed = xspeed - 0.0001f;
    else if (xspeed < 0) xspeed = xspeed + 0.0001f;
```

```
    if (yspeed > 0) yspeed = yspeed - 0.0001f;
    else if (yspeed < 0) yspeed = yspeed + 0.0001f;
    if (zspeed > 0) zspeed = zspeed - 0.0001f;
    else if (zspeed < 0) zspeed = zspeed + 0.0001f;
    if (zspeedmirror > 0) zspeedmirror = zspeedmirror - 0.0001f;
    else if (zspeedmirror < 0) zspeedmirror = zspeedmirror + 0.0001f;
    if (xmoon > 0) xmoonrot = xmoonrot - 0.0001f;
    else if (xmoon < 0) xmoon = xmoon + 0.0001f;
    if (ymoon > 0) ymoon = ymoon - 0.0001f;
    else if (ymoon < 0) ymoon = ymoon + 0.0001f;

    bmpx2 = last;
    //A switch to select what texture has to be shown on the solid sphere
    switch (type){
 case KEEPSPINNING: // animate
     glBindTexture(GL_TEXTURE_2D, texture[2]);
   break;
 case AUTOSWITCH: //switching textures
     if (teller2 == 0) teller2 = 700;
     if (teller2 % 50 == 0){
       teller2--;
       last++;
       if (last > 10) last = 0;
     }
     else teller2--;
     glBindTexture(GL_TEXTURE_2D, texture[bmpx2]);
   break;
 case MANUALSWITCH: //manually switch textures
   switch (planet) {
     case 0:  //mercury
  glBindTexture(GL_TEXTURE_2D, texture[0]);
  last = 0;
       break;
     case 1:  //venus
  glBindTexture(GL_TEXTURE_2D, texture[1]);
  last = 1;
       break;
     case 2:  //earth
  glBindTexture(GL_TEXTURE_2D, texture[2]);
  last = 2;
       break;
     case 3:  //mars
  glBindTexture(GL_TEXTURE_2D, texture[3]);
  last = 3;
       break;
     case 4:  //jupiter
  glBindTexture(GL_TEXTURE_2D, texture[4]);
  last = 4;
       break;
     case 5:  //saturn
  glBindTexture(GL_TEXTURE_2D, texture[5]);
  last = 5;
       break;
     case 6:  //uranus
  glBindTexture(GL_TEXTURE_2D, texture[6]);
  last = 6;
       break;
     case 7:  //neptune
  glBindTexture(GL_TEXTURE_2D, texture[7]);
  last = 7;
       break;
     case 8:  //pluto
  glBindTexture(GL_TEXTURE_2D, texture[8]);
  last = 8;
       break;
     case 9:  //sun
       glBindTexture(GL_TEXTURE_2D, texture[9]);
         last = 9;
       break;
     case 10:  //moon
       glBindTexture(GL_TEXTURE_2D, texture[46]);
         last = 46;
       break;
     default:
       glBindTexture(GL_TEXTURE_2D, texture[2]);
       break;
     }
```

```
    break;
  default:
      glBindTexture(GL_TEXTURE_2D, texture[last]);    // choose the texture to use.
    break;
     }
     //make sure globe fills the maximum amount of the screen.
     //glTranslatef(1.8f,0.0f,z1);              // move z units into the screen.
     glTranslatef(0.0f,0.0f,z1);              // move z units into the screen.
     glTranslatef(-twoscreen,0.0f,0.0f); // move twoscreen units to the side


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

     //turn globe into 'normal position'
     glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);

     //glRotatef(-180.0f, 0.0f, 0.0f, 1.0f);
     gluSphere(quadratic,1.3f,32,32);                  // Draw A Sphere

     //check if overlayproject needs to be drawn
     if (transparent) {
       //enable alpha testing
       glEnable(GL_ALPHA_TEST);
       //enable blending
       glEnable (GL_BLEND);
       // set kind of blening
       glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
       //select texture for second bigger transparent sphere
       if (animate){
 if (overlayproject == 42) overlayproject = 20;
 glBindTexture(GL_TEXTURE_2D, texture[overlayproject]);    // choose the texture to use.
 if (interval > 10){
   overlayproject++;
   interval = 0;
 }
       }
       else {
 glBindTexture(GL_TEXTURE_2D, texture[overlayproject]);    // choose the texture to use
       }
       interval++;
       //draw a second sphere on top of the first sphere
       gluSphere(quadratic2,1.4f,32,32);
       //disable alpha testing
       glDisable(GL_ALPHA_TEST);
       //disable blening
       glDisable (GL_BLEND);
     }

     if (moon){
       //glRotatef(-yrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
       //glRotatef(-xrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
       //glRotatef(-zrot,0.0f,0.0f,1.0f);  // Rotate on the Z Axis

       glTranslatef(0.0f,3.0f,0.0f);              // move x units into the screen.
       glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
       //glRotatef(ymoonrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
       //glRotatef(xmoonrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
       glBindTexture(GL_TEXTURE_2D, texture[47]);

       gluSphere(quadraticmoon,0.2f,32,32);          // Draw A moon
       //glRotatef(-xmoonrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
       //glRotatef(-ymoonrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
       glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
       glTranslatef(0.0,-3.0f,0.0f);              // move x units into the screen.
     }

     //set lighting
     if (!light) {
       glDisable(GL_LIGHTING);
     } else {
       glEnable(GL_LIGHTING);
     }

```

```
    glutPostRedisplay();
    // since this is double buffered, swap the buffers to display what just got drawn.
    glutSwapBuffers();
}

void DrawGLScene3()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  // Clear The Screen And The Depth Buffer
    glLoadIdentity();     // Reset The View

    yrot+=xspeed;                    // X Axis Rotation
    xrot+=yspeed;                    // Y Axis Rotation
    zrot+=zspeed;     // Z Axis Rotation
    xmoonrot+=xmoon;
    ymoonrot+=ymoon;

    // slow down after a while.
    if (xspeed > 0) xspeed = xspeed - 0.0001f;
    else if (xspeed < 0) xspeed = xspeed + 0.0001f;
    if (yspeed > 0) yspeed = yspeed - 0.0001f;
    else if (yspeed < 0) yspeed = yspeed + 0.0001f;
    if (zspeed > 0) zspeed = zspeed - 0.0001f;
    else if (zspeed < 0) zspeed = zspeed + 0.0001f;
    if (xmoonrot > 0) xmoonrot = xmoonrot - 0.0001f;
    else if (xmoonrot < 0) xmoonrot = xmoonrot + 0.0001f;
    if (ymoonrot > 0) ymoonrot = ymoonrot - 0.0001f;
    else if (ymoonrot< 0) ymoonrot = ymoonrot + 0.0001f;

    bmpx = last2;
    switch (type){
      case KEEPSPINNING: //animate
   glBindTexture(GL_TEXTURE_2D, texture[12]);
 break;
      case AUTOSWITCH: //switchingtextures
   if (teller == 0) teller = 700;
   if (teller % 100 == 0){
     teller--;
     last2++;
     if (last2 > 19) last2 = 10;
   }
   else teller--;
   glBindTexture(GL_TEXTURE_2D, texture[bmpx]);
 break;
      case MANUALSWITCH: //manually change textures
   switch (planet) {
     case 0:  //mercury
  glBindTexture(GL_TEXTURE_2D, texture[10]);
  last2 = 10;
       break;
     case 1:  //venus
         glBindTexture(GL_TEXTURE_2D, texture[11]);
  last2 = 11;
       break;
     case 2:  //earth
         glBindTexture(GL_TEXTURE_2D, texture[12]);
  last2 = 12;
       break;
     case 3:  //mars
         glBindTexture(GL_TEXTURE_2D, texture[13]);
  last2 = 13;
       break;
     case 4:  //jupiter
         glBindTexture(GL_TEXTURE_2D, texture[14]);
  last2 = 14;
       break;
     case 5:  //saturn
         glBindTexture(GL_TEXTURE_2D, texture[15]);
  last2 = 15;
       break;
     case 6:  //uranus
         glBindTexture(GL_TEXTURE_2D, texture[16]);
  last2 = 16;
       break;
     case 7:  //neptune
         glBindTexture(GL_TEXTURE_2D, texture[17]);
  last2 = 17;
```

```
        break;
      case 8:  //pluto
          glBindTexture(GL_TEXTURE_2D, texture[18]);
  last2 = 18;
        break;
      case 9:  //sun
          glBindTexture(GL_TEXTURE_2D, texture[19]);
  last2 = 19;
        break;
      default:
          glBindTexture(GL_TEXTURE_2D, texture[12]);
        break;
         }
         break;

       default:
   glBindTexture(GL_TEXTURE_2D, texture[last2]);   // choose the texture to use.
 break;
    //manually change textures
    }

    //glTranslatef(1.8f,0.0f,z2);            // move z units into the screen, and 1.8 units to the side.
    glTranslatef(0.0f,0.0f,z2);            // move z units into the screen, and 1.8 units to the side.
    glRotatef(-yrot,1.0f,0.0f,1.0f);  // Rotate On The X Axis
    glRotatef(xrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
    glRotatef(-zrot,0.0f,0.0f,1.0f);  // Rotate on the Z Axis

    glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
    gluSphere(quadratic,1.3f,32,32);                // Draw A Sphere

    // first check if needs to be transparent
    if (transparent) {
      //enable alpha testing
      glEnable(GL_ALPHA_TEST);
      //enable the blend function
      glEnable (GL_BLEND);
      // set blending function
      glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
      //select texture for second bigger transparent sphere
      if (animate){
 if (overlayproject == 42) overlayproject = 20;
 glBindTexture(GL_TEXTURE_2D, texture[overlayproject]);   // choose the texture to use.
 overlayproject++;
      }
      else {
 glBindTexture(GL_TEXTURE_2D, texture[43]);   // choose the texture to use
      }
      //draw a second sphere on top of the first sphere
      gluSphere(quadratic2,1.4f,32,32);
      //disable alpha testing
      glDisable(GL_ALPHA_TEST);
      //disable blending
      glDisable (GL_BLEND);
    }

    if (moon){
      glRotatef(yrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
      glRotatef(-xrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
      glRotatef(zrot,0.0f,0.0f,1.0f);  // Rotate on the Z Axis

      glTranslatef(0.0f,-2.0f,0.0f);            // move x units into the screen.
      glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
      glRotatef(ymoonrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
      glRotatef(-xmoonrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
      glBindTexture(GL_TEXTURE_2D, texture[9]);

      gluSphere(quadraticmoon,0.2f,32,32);          // Draw A moon
      glRotatef(-xmoonrot,1.0f,0.0f,0.0f);  // Rotate On The X Axis
      glRotatef(ymoonrot,0.0f,1.0f,0.0f);  // Rotate On The Y Axis
      glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
      glTranslatef(0.0,2.0f,0.0f);            // move x units into the screen.
    }

    //set lighting
    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
```

```
 glBegin(GL_TRIANGLE_FAN);
 glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
 glTexCoord2f(0, 0); glVertex3f(640,0,0);
 glTexCoord2f(1, 0); glVertex3f(1280,0,0);
 glTexCoord2f(1, 1); glVertex3f(1280,480,0);
 glTexCoord2f(0, 1); glVertex3f(640,480,0);
 glEnd();

 glutSwapBuffers();
}

/* The function called when our window is resized  */
void ReSizeGLScene2(GLsizei Width, GLsizei Height)
{
    if (Height==0){  // Prevent A Divide By Zero If The Window Is Too Small
 Height=1;
    }

    glViewport(0, 0, Width, Height); // Reset The Current Viewport And Perspective Transformation

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    gluPerspective(45.0f, (GLfloat)Width / (GLfloat)Height, 0.1f, 100.0f);
    glMatrixMode(GL_MODELVIEW);
}

void ReSizeGLScene(int Width, int Height){
   if (Height==0){  // Prevent A Divide By Zero If The Window Is Too Small
 Height=1;
    }
  glViewport(0,0,Width,Height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho (0, 1280, 480, 0, -1.0f, 1.0f);
  glMatrixMode(GL_MODELVIEW);
}

void InitGL(int Width, int Height)
{
 glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
 glClearDepth(1.0);
 glDepthFunc(GL_LESS);
 glDepthMask(GL_FALSE);
 glDisable(GL_DEPTH_TEST);
 glDisable(GL_BLEND);
 glDisable(GL_ALPHA_TEST);
 glEnable(GL_TEXTURE_2D);
 glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
 glShadeModel(GL_FLAT);

 glGenTextures(1, &gl_depth_tex);
 glBindTexture(GL_TEXTURE_2D, gl_depth_tex);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

 glGenTextures(1, &gl_rgb_tex);
 glBindTexture(GL_TEXTURE_2D, gl_rgb_tex);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

 ReSizeGLScene(Width, Height);
}

/* A general OpenGL initialization function.  Sets all of the initial parameters. */
void InitGL2(GLsizei Width, GLsizei Height) // We call this right after our OpenGL window is created.
{
    LoadGLTextures();     // Load the textures
    glEnable(GL_TEXTURE_2D);    // Enable texture mapping

    glClearColor(0.0f, 0.0f, 0.0f, 0.0f); // This Will Clear The Background Color To Black
    glClearDepth(1.0);     // Enables Clearing Of The Depth Buffer
    glDepthFunc(GL_LESS);    // The Type Of Depth Test To Do
    glEnable(GL_DEPTH_TEST);    // Enables Depth Testing
```

```
    glShadeModel(GL_SMOOTH);    // Enables Smooth Color Shading
    glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();     // Reset The Projection Matrix

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

    glMatrixMode(GL_MODELVIEW);

    // set up light number 1.
    glLightfv(GL_LIGHT1, GL_AMBIENT, LightAmbient);    // add lighting. (ambient)
    glLightfv(GL_LIGHT1, GL_DIFFUSE, LightDiffuse);    // add lighting. (diffuse).
    glLightfv(GL_LIGHT1, GL_POSITION,LightPosition);   // set light position.
    glEnable(GL_LIGHT1);                               // turn light 1 on.

    /* setup blending */
    glColor4f(1.0f, 1.0f, 1.0f, 0.25);
    quadratic=gluNewQuadric();                    // Create A Pointer To The Quadric Object ( NEW )
    quadratic2=gluNewQuadric();                    // Create A Pointer To The Quadric Object ( NEW )
    quadraticmoon=gluNewQuadric();   //moon

    // Can also use GLU_NONE, GLU_FLAT
    gluQuadricNormals(quadratic, GLU_SMOOTH);    // Create Smooth Normals
    gluQuadricTexture(quadratic, GL_TRUE);       // Create Texture Coords ( NEW )

    gluQuadricNormals(quadratic2, GLU_SMOOTH);    // Create Smooth Normals
    gluQuadricTexture(quadratic2, GL_TRUE);       // Create Texture Coords ( NEW )

    gluQuadricNormals(quadraticmoon, GLU_SMOOTH);    // Create Smooth Normals
    gluQuadricTexture(quadraticmoon, GL_TRUE);       // Create Texture Coords ( NEW )
}
```

A.4 imageload.c

load the textures into the program.

```

#include <stdio.h>

#include "imageload.h"
#include "libfreenect.h"

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

```
  filenamearray[23] = "./bin/Data/texture/overlay/4.bmp";
  filenamearray[24] = "./bin/Data/texture/overlay/5.bmp";
  filenamearray[25] = "./bin/Data/texture/overlay/6.bmp";
  filenamearray[26] = "./bin/Data/texture/overlay/7.bmp";
  filenamearray[27] = "./bin/Data/texture/overlay/8.bmp";
  filenamearray[28] = "./bin/Data/texture/overlay/9.bmp";
  filenamearray[29] = "./bin/Data/texture/overlay/10.bmp";
  filenamearray[30] = "./bin/Data/texture/overlay/11.bmp";
  filenamearray[31] = "./bin/Data/texture/overlay/12.bmp";
  filenamearray[32] = "./bin/Data/texture/overlay/13.bmp";
  filenamearray[33] = "./bin/Data/texture/overlay/14.bmp";
  filenamearray[34] = "./bin/Data/texture/overlay/15.bmp";
  filenamearray[35] = "./bin/Data/texture/overlay/16.bmp";
  filenamearray[36] = "./bin/Data/texture/overlay/17.bmp";
  filenamearray[37] = "./bin/Data/texture/overlay/18.bmp";
  filenamearray[38] = "./bin/Data/texture/overlay/19.bmp";
  filenamearray[39] = "./bin/Data/texture/overlay/20.bmp";
  filenamearray[40] = "./bin/Data/texture/overlay/21.bmp";
  filenamearray[41] = "./bin/Data/texture/overlay/22.bmp";
  filenamearray[42] = "./bin/Data/texture/overlay/23.bmp";
  filenamearray[43] = "./bin/Data/texture/overlay/test1.bmp";
  filenamearray[44] = "./bin/Data/texture/overlay/test2.bmp";
  filenamearray[45] = "./bin/Data/texture/old/crate.bmp";
  filenamearray[46] = "./bin/Data/texture/moon2.bmp";
  filenamearray[47] = "./bin/Data/texture/moon.bmp";
}

/*
 *  Load texture into memory
 *  LoadGLTextures runs throught the filenamearray and allocates memory
 *  for all the images. And than makes textures out of them
 */
void LoadGLTextures() {
  int x = 0;
  // Allocate space for texture
  while (x < MAX_IMAGES){
    image[x] = (Image *) malloc(sizeof(Image));
    if (image[x] == NULL) {
   printf("Error allocating space for image %d \n", x);
   exit(0);
    }
    //printf("trying to allocate space for %s \n", filenamearray[x]);
    if (!ImageLoad(filenamearray[x], image[x])) {
     exit(1);
    }
    // create Texture
    glGenTextures(3, &texture[x]);
    // texture 2 (linear scaling)
    glBindTexture(GL_TEXTURE_2D, texture[x]);   // 2d texture (x and y size)
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR); // scale linearly when image bigger than texture
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR); // scale linearly when image smalled than texture
    glTexImage2D(GL_TEXTURE_2D, 0, 3, image[x]->sizeX, image[x]->sizeY, 0, GL_RGB, GL_UNSIGNED_BYTE, image[x]->data);
    // 2d texture, 3 colors, width, height, RGB in that order, byte data, and the data.
    gluBuild2DMipmaps(GL_TEXTURE_2D, 3, image[x]->sizeX, image[x]->sizeY, GL_RGB, GL_UNSIGNED_BYTE, image[x]->data);
    x++;
  }
}

/*
 *  Load an image from .BMP file
 *  reads image header to see if image is valid
 *  has to be 24 bit bmp
 */
int ImageLoad(char *filename, Image *image){
    FILE *file;
    unsigned long size;                 // size of the image in bytes.
    unsigned long i;                    // standard counter.
    unsigned short int planes;          // number of planes in image (must be 1)
    unsigned short int bpp;             // number of bits per pixel (must be 24)
    char temp;                          // used to convert bgr to rgb color.


    // Make sure the file exists
```

```
    if ((file = fopen(filename, "rb"))==NULL){
 printf("File Not Found : %s\n",filename);
 return 0;
    }

    if (file){
 //if file exist get header
 struct INFOHEADER bmpheader;
 int width,height;
 //printf("\nYeah..! Image opened\n");
 fread(&bmpheader,sizeof(bmpheader),1,file);
 //printf("\nImage width=%d\n",bmpheader.width);
 //printf("\nImage Height=%d\n",bmpheader.height);
 width = bmpheader.width;
 height = bmpheader.height;
 planes = bmpheader.planes;
 bpp = bmpheader.bits;

 // get the size (assuming 24 bpp)
 size = bmpheader.imagesize;
 //printf("width * height equals %lu \n", image->sizeX * image->sizeY * 3);
 //printf("size of image equals %lu \n", size);

 //set size in image class
 image->sizeX = width;
 image->sizeY = height;

 //check if image is valid
 if (planes != 1) {
  printf("Planes from %s is not 1: %u\n", filename, planes);
  return 0;
 }

 if (bpp != 24) {
  printf("Bpp from %s is not 24: %u\n", filename, bpp);
  //return 0;
 }


 /* move to start of image data */
 rewind(file);
 fseek(file, (long)bmpheader.offset_to_start_of_image_data_in_bytes, SEEK_SET);

 // Allocate memory for the data
 image->data = (char *) malloc(size);
 if (image->data == NULL) {
  printf("Error allocating memory for colour-corrected image data");
  return 0;
 }

 // Read the data and set in allocated memory space
 if ((i = fread(image->data,size,1,file)) != 1) {
  printf("Error reading image data from %s.\n", filename);
  return 0;
 }

 // reverse all of the colours bgr => rgb)
 for (i=0;i<size;i+=3) {
  temp = image->data[i];
  image->data[i] = image->data[i+2];
  image->data[i+2] = temp;
 }
    }
    // Thats all folks
    return 1;

}
```

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':
```

```
if (type != MANUALSWITCH) type = MANUALSWITCH;
else type = DEFAULTVIEW;
      break;
    case '+':
planet++;
if (planet == 11) planet = 0;
      break;
    case '-':
planet--;
if (planet == -1) planet = 10;
      break;
    case KEY_PAD_UP: //numpad up 8
xspeed += 0.01f;
      break;
    case KEY_PAD_DOWN: //numpad down 2
xspeed -= 0.01f;
      break;
    case KEY_PAD_LEFT: //numpad left 4
yspeed -= 0.01f;
      break;
    case KEY_PAD_RIGHT: //numpad right 6
yspeed += 0.01f;
      break;
    case KEY_PAD_MIDDLE: //numpad centre 5
yspeed = 0.0f;
xspeed = 0.0f;
zspeed = 0.0f;
zspeedmirror = 0.0f;
type = DEFAULTVIEW;
      break;
    case KEY_PAD_LEFT_UP: //numpad 7
zspeed -=0.01f;
      break;
    case KEY_PAD_RIGHT_UP: //numpad 9
 zspeed +=0.01f;
      break;
    case KEY_PAD_LEFT_DOWN: //numpad 9
 zspeedmirror -=0.01f;
      break;
    case KEY_PAD_RIGHT_DOWN: //numpad 9
 zspeedmirror +=0.01f;
      break;
    case 'w':
freenect_angle++;
if (freenect_angle > 30) {
  freenect_angle = 30;
}
freenect_set_tilt_degs(f_dev,freenect_angle);
      break;
    case 'x':
freenect_angle = 0;
freenect_set_tilt_degs(f_dev,freenect_angle);
      break;
    case 'f':
if (requested_format == FREENECT_VIDEO_IR_8BIT)
  requested_format = FREENECT_VIDEO_RGB;
else if (requested_format == FREENECT_VIDEO_RGB)
  requested_format = FREENECT_VIDEO_YUV_RGB;
else
  requested_format = FREENECT_VIDEO_IR_8BIT;
freenect_set_tilt_degs(f_dev,freenect_angle);
      break;
    case 's':
        freenect_angle--;
if (freenect_angle < -30) {
  freenect_angle = -30;
}
freenect_set_tilt_degs(f_dev,freenect_angle);
      break;
    case ']':
if (cycle == 0) freenect_set_led(f_dev,LED_OFF);
if (cycle == 1) freenect_set_led(f_dev,LED_GREEN);
if (cycle == 2) freenect_set_led(f_dev,LED_RED);
if (cycle == 3) freenect_set_led(f_dev,LED_YELLOW);
if (cycle == 4) freenect_set_led(f_dev,LED_BLINK_GREEN);
if (cycle == 5) freenect_set_led(f_dev,LED_BLINK_GREEN);
if (cycle == 6) freenect_set_led(f_dev,LED_BLINK_RED_YELLOW);
```

```
 if (cycle == 1) freenect_set_led(f_dev,LED_GREEN);
 cycle++;
 if (cycle == 7)cycle = 0;
      break;
    case 'm':
    case 'M':
        //moon = !moon;
      break;
    default:
        printf ("Key %d pressed. No action there yet.\n", key);
      break;
    }
}

/* The function called whenever a normal key is pressed. */
void specialKeyPressed(int key, int x, int y){
    /* avoid thrashing this procedure */
    usleep(100);

    switch (key) {
    case GLUT_KEY_PAGE_UP: // move the cube into the distance.
  z1-=0.02f;
 break;

    case GLUT_KEY_PAGE_DOWN: // move the cube closer.
  z1+=0.02f;
 break;

    case GLUT_KEY_UP: // decrease x rotation speed;
   z2-=0.02f;
 break;

    case GLUT_KEY_DOWN: // increase x rotation speed;
   z2+=0.02f;
 break;

    case GLUT_KEY_LEFT: // decrease y or z rotation speed;
   xmoon+=0.01f;
 break;

    case GLUT_KEY_RIGHT: // increase y of z rotation speed;
   xmoon-=0.01f;
 break;

    case GLUT_KEY_END:
   yspeed = 0.0f;
   xspeed = 0.0f;
   zspeed = 0.0f;
   zspeedmirror = 0.0f;
   yrot = 0;
   xrot = 0;
   zrot = 0;
   zrotmirror = 0;
   xmoon = 0.0f;
   ymoon = 0.0f;
   type = DEFAULTVIEW;
   animate = GL_FALSE;
 break;

    case GLUT_KEY_HOME:
 if (type != KEEPSPINNING){
   type = KEEPSPINNING;
          animate =!animate;
 }
 else {
   type = DEFAULTVIEW;
   animate =!animate;
 }
 break;

     default:
 break;
    }
}

void MouseEvent(int button, int state, int x, int y){

```

```
    key_modifiers=glutGetModifiers();
    if (button == GLUT_LEFT_BUTTON) {
        if (state == GLUT_DOWN) {
            moving = 1;
            start_x = x;
            start_y = y;
        }
        if (state == GLUT_UP) {
            moving = 0;
        }
    }
    glutPostRedisplay();
}

void MouseMotion(int x, int y){
    static double acc;

    if(moving){
        if( key_modifiers & GLUT_ACTIVE_CTRL ) acc=0.02; else acc=1.0;
        if(!animate){
            yrot -=  (double)(y-start_y)*acc;
            xrot +=  (double)(x-start_x)*acc;
        } else {
            yspeed = (double)(x-start_x)*0.05f;
            xspeed = (double)(y-start_y)*0.05f;
        }
        start_x = x;
        start_y = y;
    }
    glutPostRedisplay();
}
```

A.6 kinect.c

Kinect code, to register input

```

#include <stdio.h>
#include <stdlib.h>

#include "libfreenect.h"
#include "kinect.h"

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

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

void kinect_get_state()
```

```
{

}

void getMovement(){
  static double acc;
  acc = 0.3;
  int reset;
  //printf("moving: %i", moving);
  if (!moving){
    if (meanx == meanxoud && meany == meanyoud && meanx !=0 && meany !=0){
      if (!animate && teller3 > 20){
 teller3 = 0;
 moving = 1;
      }
      else if (teller3 > 10){
        teller3 = 0;
 moving = 1;
      }
      teller3++;
      //printf("teller3: %i", teller3);
    }
  }
  if (moving){
    if (meanx == 0 && meany == 0){
      if (teller3 > 40){
 teller3 = 0;
 moving = 0;
      }
      else
 teller3++;
    }
    else if(!animate){
      xrot -=  (double)(meany-meanyoud*2)*acc+2;
      yrot +=  (double)(meanx-meanxoud*2)*acc+2;
    }
    if (animate){
      if (meanxoud != 0 && meanx !=0){
 if (meanxoud - meanx < 0){
   xspeed += 0.2f;
   reset = 0;
 }
 else if (meanxoud - meanx > 0){
   xspeed -= 0.2f;
   reset = 0;
 }
      }
      if(meanyoud != 0 && meany != 0){
   if (meanyoud - meany > 0){
     yspeed -= 0.2f;
     reset = 0;
   }
   else if (meanyoud - meany < 0){
     yspeed += 0.2f;
     reset = 0;
   }
      }
      //if (meanxoud == meanx && meanyoud == meany){
 //if (reset == 5){
   //yspeed -= 0;
   //xspeed -= 0;
   //reset = 0;
 //}
 //else reset++;
      //}
    }
    if ((meanx == 0 || meany == 0) && (meanyoud == 0 || meanxoud == 0) && !animate){
      //stop rotation
      xspeed = 0;
      yspeed = 0;
      moving = 0;
    }
  }
  meanxoud = meanx;
  meanyoud = meany;
}

```

```
void emptyarray()
{
    int i, j;
    //empty arrays
  for (i = 0; i < 240; i++){
    for (j = 0; j < 320; j++){
      smallblocks[i][j] = 0;
    }
  }
  for (i = 0; i < 120; i++){
    for (j = 0; j < 160; j++){
      mediumblocks[i][j] = 0;
    }
  }
  for (i = 0; i < 60; i++){
    for (j = 0; j < 80; j++){
      largeblocks[i][j] = 0;
    }
  }
  for (i = 0; i < 30; i++){
    for (j = 0; j < 40; j++){
      bigblocks[i][j] = 0;
    }
  }
  for (i = 0; i < 15; i++){
    for (j = 0; j < 20; j++){
      hugeblocks[i][j] = 0;
    }
  }
  for (i = 0; i < 15; i++){
    for (j = 0; j < 15; j++){
      blob[i][j] = 0;
    }
  }
}


void getBlobs(){
  printf("frame++\n");
  int i, j;
  int temp = 0;
  int count= 0;


  for (i = 0; i < 480; i++){
    for (j = 0; j < 640; j++){
      if (loc[i][j] != 1) count++;
    }
  }
  printf("count= %i\n", count);
  count = 0;

  // first make square blocks
  for (i = 0; i < 240; i++){
    for (j = 0; j < 320; j++){
      if (loc[2*i][2*j] != 1){
 temp++;
      }
      if (loc[2*i+1][2*j] != 1){
 temp++;
      }
      if (loc[2*i][2*j+1] != 1){
 temp++;
      }
      if (loc[2*i+1][2*j+1] != 1){
 temp++;
      }
      if (temp > 1){
 smallblocks[i][j] = 1;
 temp = 0;
      }
      else {
 smallblocks[i][j] = 0;
 temp = 0;
      }
    }
  }
```

```
 for (i = 0; i < 240; i++){
   for (j = 0; j < 320; j++){
     if (smallblocks[i][j] == 1) count++;
   }
 }
 printf("smallcount= %i\n", count);
 count = 0;

 //make the blocks bigger
 temp = 0;
 for (i = 0; i < 120; i++){
   for (j = 0; j < 160; j++){
     if (smallblocks[2*i][2*j] == 1){
temp++;
     }
     if (smallblocks[2*i+1][2*j] == 1){
temp++;
     }
     if (smallblocks[2*i][2*j+1] == 1){
temp++;
     }
     if (smallblocks[2*i+1][2*j+1] == 1){
temp++;
     }
     if (temp > 1){
mediumblocks[i][j] = 1;
temp = 0;
     }
     else {
mediumblocks[i][j] = 0;
temp = 0;
     }
   }
 }
 for (i = 0; i < 120; i++){
   for (j = 0; j < 160; j++){
     if (mediumblocks[i][j] == 1) count++;
   }
 }
 printf("mediumcount= %i\n", count);
 count = 0;

 //make the blocks bigger
 temp = 0;
 for (i = 0; i < 60; i++){
   for (j = 0; j < 80; j++){
     if (mediumblocks[2*i][2*j] == 1){
temp++;
     }
     if (mediumblocks[2*i+1][2*j] == 1){
temp++;
     }
     if (mediumblocks[2*i][2*j+1] == 1){
temp++;
     }
     if (mediumblocks[2*i+1][2*j+1] == 1){
temp++;
     }
     if (temp > 1){
largeblocks[i][j] = 1;
temp = 0;
     }
     else{
largeblocks[i][j] = 0;
temp = 0;
     }
   }
 }
 for (i = 0; i < 60; i++){
   for (j = 0; j < 80; j++){
     if (largeblocks[i][j] == 1) count++;
   }
 }
 printf("largecount= %i\n", count);
 count = 0;

 //make the blocks bigger
```

```
 temp = 0;
 for (i = 0; i < 30; i++){
   for (j = 0; j < 40; j++){
     if (largeblocks[2*i][2*j] == 1){
temp++;
     }
     if (largeblocks[2*i+1][2*j] == 1){
temp++;
     }
     if (largeblocks[2*i][2*j+1] == 1){
temp++;
     }
     if (largeblocks[2*i+1][2*j+1] == 1){
temp++;
     }
     if (temp > 2){
bigblocks[i][j] = 1;
temp = 0;
     }
     else{
bigblocks[i][j] = 0;
temp = 0;
     }
   }
 }
 for (i = 0; i < 30; i++){
   for (j = 0; j < 40; j++){
     if (bigblocks[i][j] == 1) count++;
   }
 }
 printf("bigcount= %i\n", count);
 count = 0;

 // big blocks
 temp = 0;
 for (i = 0; i < 15; i++){
   for (j = 0; j < 20; j++){
     if (bigblocks[2*i][2*j] == 1){
temp++;
     }
     if (bigblocks[2*i+1][2*j] == 1){
temp++;
     }
     if (bigblocks[2*i][2*j+1] == 1){
temp++;
     }
     if (bigblocks[2*i+1][2*j+1] == 1){
temp++;
     }
     if (temp > 1){
hugeblocks[i][j] = 1;
temp = 0;
     }
     else{
hugeblocks[i][j] = 0;
temp = 0;
     }
   }
 }
 for (i = 0; i < 15; i++){
   for (j = 0; j < 20; j++){
     if (hugeblocks[i][j] == 1) count++;
   }
 }
 printf("hugecount= %i\n", count);
 count = 0;

 //now we throw away the left most 2 bits and the rightmost 3 bits
 for(i = 0; i < 15; i++){
   for (j = 0; j < 15; j++){
     blob[i][j] = hugeblocks[i+2][j];
   }
 }
 int row[15];
 int column[16];
 for (i = 0; i < 15; i++){
   row[i]= 0;
```

```
  }
  for (j = 0; j < 16; j++){
    column[j]=0;
  }

  for (i = 0;i < 15; i++){
    for (j = 0; j < 16; j ++){
      if (hugeblocks[i][j] == 1){
row[i] = 1;
column[j] = 1;
      }
    }
  }

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

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

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

A.7 kubus2.c

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

```

#include <stdio.h>
#include <stdlib.h>

#include "libfreenect.h"
#include "kinect.h"

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

```
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;
```

```
    }
        }
        //if (meanxoud == meanx && meanyoud == meany){
  //if (reset == 5){
    //yspeed -= 0;
    //xspeed -= 0;
    //reset = 0;
  //}
  //else reset++;
        //}
      }
      if ((meanx == 0 || meany == 0) && (meanyoud == 0 || meanxoud == 0) && !animate){
        //stop rotation
        xspeed = 0;
        yspeed = 0;
        moving = 0;
      }
    }
    meanxoud = meanx;
    meanyoud = meany;
}

void emptyarray()
{
      int i, j;
      //empty arrays
    for (i = 0; i < 240; i++){
      for (j = 0; j < 320; j++){
        smallblocks[i][j] = 0;
      }
    }
    for (i = 0; i < 120; i++){
      for (j = 0; j < 160; j++){
        mediumblocks[i][j] = 0;
      }
    }
    for (i = 0; i < 60; i++){
      for (j = 0; j < 80; j++){
        largeblocks[i][j] = 0;
      }
    }
    for (i = 0; i < 30; i++){
      for (j = 0; j < 40; j++){
        bigblocks[i][j] = 0;
      }
    }
    for (i = 0; i < 15; i++){
      for (j = 0; j < 20; j++){
        hugeblocks[i][j] = 0;
      }
    }
    for (i = 0; i < 15; i++){
      for (j = 0; j < 15; j++){
        blob[i][j] = 0;
      }
    }
}


void getBlobs(){
    printf("frame++\n");
    int i, j;
    int temp = 0;
    int count= 0;


    for (i = 0; i < 480; i++){
      for (j = 0; j < 640; j++){
        if (loc[i][j] != 1) count++;
      }
    }
    printf("count= %i\n", count);
    count = 0;

    // first make square blocks
    for (i = 0; i < 240; i++){
      for (j = 0; j < 320; j++){
```

```
      if (loc[2*i][2*j] != 1){
temp++;
      }
      if (loc[2*i+1][2*j] != 1){
temp++;
      }
      if (loc[2*i][2*j+1] != 1){
temp++;
      }
      if (loc[2*i+1][2*j+1] != 1){
temp++;
      }
      if (temp > 1){
smallblocks[i][j] = 1;
temp = 0;
      }
      else {
smallblocks[i][j] = 0;
temp = 0;
      }
    }
  }
  for (i = 0; i < 240; i++){
    for (j = 0; j < 320; j++){
      if (smallblocks[i][j] == 1) count++;
    }
  }
  printf("smallcount= %i\n", count);
  count = 0;

  //make the blocks bigger
  temp = 0;
  for (i = 0; i < 120; i++){
    for (j = 0; j < 160; j++){
      if (smallblocks[2*i][2*j] == 1){
temp++;
      }
      if (smallblocks[2*i+1][2*j] == 1){
temp++;
      }
      if (smallblocks[2*i][2*j+1] == 1){
temp++;
      }
      if (smallblocks[2*i+1][2*j+1] == 1){
temp++;
      }
      if (temp > 1){
mediumblocks[i][j] = 1;
temp = 0;
      }
      else {
mediumblocks[i][j] = 0;
temp = 0;
      }
    }
  }
  for (i = 0; i < 120; i++){
    for (j = 0; j < 160; j++){
      if (mediumblocks[i][j] == 1) count++;
    }
  }
  printf("mediumcount= %i\n", count);
  count = 0;

  //make the blocks bigger
  temp = 0;
  for (i = 0; i < 60; i++){
    for (j = 0; j < 80; j++){
      if (mediumblocks[2*i][2*j] == 1){
temp++;
      }
      if (mediumblocks[2*i+1][2*j] == 1){
temp++;
      }
      if (mediumblocks[2*i][2*j+1] == 1){
temp++;
      }
```

```
      if (mediumblocks[2*i+1][2*j+1] == 1){
temp++;
      }
      if (temp > 1){
largeblocks[i][j] = 1;
temp = 0;
      }
      else{
largeblocks[i][j] = 0;
temp = 0;
      }
    }
  }
  for (i = 0; i < 60; i++){
    for (j = 0; j < 80; j++){
      if (largeblocks[i][j] == 1) count++;
    }
  }
  printf("largecount= %i\n", count);
  count = 0;

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

  // big blocks
  temp = 0;
  for (i = 0; i < 15; i++){
    for (j = 0; j < 20; j++){
      if (bigblocks[2*i][2*j] == 1){
temp++;
      }
      if (bigblocks[2*i+1][2*j] == 1){
temp++;
      }
      if (bigblocks[2*i][2*j+1] == 1){
temp++;
      }
      if (bigblocks[2*i+1][2*j+1] == 1){
temp++;
      }
      if (temp > 1){
hugeblocks[i][j] = 1;
temp = 0;
      }
      else{
hugeblocks[i][j] = 0;
```

```
 temp = 0;
        }
      }
    }
    for (i = 0; i < 15; i++){
      for (j = 0; j < 20; j++){
        if (hugeblocks[i][j] == 1) count++;
      }
    }
    printf("hugecount= %i\n", count);
    count = 0;

    //now we throw away the left most 2 bits and the rightmost 3 bits
    for(i = 0; i < 15; i++){
      for (j = 0; j < 15; j++){
        blob[i][j] = hugeblocks[i+2][j];
      }
    }
    int row[15];
    int column[16];
    for (i = 0; i < 15; i++){
      row[i]= 0;
    }
    for (j = 0; j < 16; j++){
      column[j]=0;
    }

    for (i = 0;i < 15; i++){
      for (j = 0; j < 16; j ++){
        if (hugeblocks[i][j] == 1){
 row[i] = 1;
 column[j] = 1;
        }
      }
    }

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

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

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

A.8 test.c

Program to check if the kinect input works

```
#include <stdio.h>
#include <pthread.h>
#include <math.h>
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>

#include "libfreenect.h"
#include "threadfunc.h"
#include "drawX.h"



void *gl_threadfunc2(void *arg)
{
  glutInit(&g_argc, g_argv);
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |  GLUT_DEPTH );

  glutInitWindowSize(1280, 480);
  glutInitWindowPosition(0, 0);
  window3 = glutCreateWindow("LibFreenect");
  glutDisplayFunc(&DrawGLScene);
  glutIdleFunc(&DrawGLScene);
  glutReshapeFunc(&ReSizeGLScene);
  /* Register the function called when the keyboard is pressed. */
  glutKeyboardFunc(&keyPressed);
  /* Register the function called when special keys (arrows, page down, etc) are pressed
      . */
  glutSpecialFunc(&specialKeyPressed);
  InitGL(1280, 480);

  glutMainLoop();
  return NULL;
}

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

  printf("Kinect camera test\n");

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

  g_argc = argc;
  g_argv = argv;

  if (freenect_init(&f_ctx, NULL) < 0) {
   printf("freenect_init() failed\n");
   return 1;
  }

  freenect_set_log_level(f_ctx, FREENECT_LOG_DEBUG);

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

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

  if (nr_devices < 1)
   return 1;
```

```

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

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

  // OS X requires GLUT to run on the main thread
  gl_threadfunc2(NULL);

  return 0;

}
```

A.9 defines.h

Code file with all defines

```

#ifndef DEFINES_H
#define DEFINES_H

#define FREENECT_COUNTS_PER_G 819 /**< Ticks per G for accelerometer as set per http://www.kionix.com/Product%20Sheets/KXSD9%20Product%20Brief.pdf */
#define TRUE  1
#define FALSE 0
#define ESCAPE 27
#define PAGE_UP 73
#define PAGE_DOWN 81
#define UP_ARROW 72
#define DOWN_ARROW 80
#define LEFT_ARROW 75
#define RIGHT_ARROW 77
#define KEY_DEL 127
#define KEY_PAD_UP 56
#define KEY_PAD_DOWN 50
#define KEY_PAD_LEFT 52
#define KEY_PAD_RIGHT 54
#define KEY_PAD_MIDDLE 53
#define KEY_PAD_LEFT_UP 55
#define KEY_PAD_LEFT_DOWN 49
#define KEY_PAD_RIGHT_UP 57
#define KEY_PAD_RIGHT_DOWN 51
#define MAX_IMAGES 48

#define KEEPSPINNING 0
#define AUTOSWITCH 1
#define MANUALSWITCH 2
#define DEFAULTVIEW 4

#define KUBUS 0
#define GLOBES 1
#define KINECTTEST 2


// change DEBUG to 1 if you want to see the kinect feed in Glview
#define DEBUG 0

#endif
```

A.10 drawX.h

Code of drawX.h

```

#ifndef DRAW_H
#define DRAW_H
#include "libfreenect.h"

int window, window2, window3;  // The numbers of our GLUT windows
extern GLfloat  ymoon;
extern GLfloat  xmoon;
extern GLfloat  ymoonrot;
extern GLfloat  xmoonrot;
extern GLfloat  yrot;   // X Rotation
extern GLfloat  xrot;   // Y Rotation
extern GLfloat  zrot;   // Z Rotation
extern GLfloat  zrotmirror;   // Z Rotation
extern GLfloat  xspeed;   // x rotation speed
extern GLfloat  yspeed;   // y rotation speed
extern GLfloat  zspeed;   // Z rotation speed
extern GLfloat  zspeedmirror;   // Z rotation speed
extern GLfloat  z1;   // depth into the screen.
extern GLfloat  z2;   // depth into the screen.
extern GLshort transparent;
extern GLshort moon;

extern int type;   // 0 for animate, 1 for switching textures, 2 for manually change
extern int teller;   // counter to control the pace of changing textures.
extern int teller2;   // counter to control the pace of changing textures.
extern int last;   // last shown was the earth
extern int last2;   // last shown was the earth
extern int teller3;
extern int planet;   // for manually changing the planets a counter to keep track of
      which planet we look at.
extern int bmpx;   // startnr of texture in second window
extern int bmpx2;   // startnr of texture in first window
extern int cycle;   // variable to keep track of wich led color the kinect must show


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


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


#endif
```

A.11 imageload.h

Code of imageload.h

```

#ifndef IMAGELOAD_H
#define IMAGELOAD_H

#include "libfreenect.h"

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

/* Image type - contains height, width and data */
struct Image {
 unsigned long sizeX;
 unsigned long sizeY;
 char *data;
```

```
};
typedef struct Image Image;

/*
 * Struct infoheader
 * found at
 *   http://iamfeelinlucky.blogspot.nl/2009/09/c-code-to-read-bmp-header-information.html
 */
struct INFOHEADER
{
        unsigned short int signature;
        unsigned int size;
        unsigned short int reserved1;
        unsigned short int reserved2;
        int offset_to_start_of_image_data_in_bytes;
        int size_of_BITMAPINFOHEADER;
        int width,height;
        unsigned short int planes;
        unsigned short int bits;
        unsigned int compression;
        unsigned int imagesize;
        int xresolution,yresolution;
        unsigned int ncolours;
        unsigned int importantcolours;
}__attribute__((__packed__));
typedef struct INFOHEADER INFOHEADER;

int location[640][480];

Image *image[MAX_IMAGES]; // variable for the images

void FillArrayImageNames();
int ImageLoad(char *filename, Image *image);
void LoadGLTextures();


#endif
```

A.12 interact.h

Code of interact.h

```

#ifndef INTERACT_H
#define INTERACT_H
#include "libfreenect.h"

int light;     // Lighting ON/OFF
int update;
int meanxoud;
int meanyoud;
int meanx;
int meany;

extern GLboolean animate; // When true movement continues till it reguarly slowed down
int     moving, start_x, start_y; // variables to keep track of mousemovement
int      key_modifiers;   // variable to check if there is movement

extern int   type;   // 0 for animate, 1 for switching textures, 2 for manually change
extern int   teller;   // counter to control the pace of changing textures.
extern int   teller2;   // counter to control the pace of changing textures.
extern int   last;   // last shown was the earth
extern int   last2;   // last shown was the earth

extern int   planet;   // for manually changing the planets a counter to keep track of
     which planet we look at.
extern int   bmpx;   // startnr of texture in second window
extern int   bmpx2;   // startnr of texture in first window

void keyPressed(unsigned char key, int x, int y);
void specialKeyPressed(int key, int x, int y);
void MouseEvent(int button, int state, int x, int y);
void MouseMotion(int x, int y);
```

```


#endif
```

A.13 kinect.h

Code of kinect.h

```

#ifndef KINECT_H
#define KINECT_H

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

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

  void kinect_get_position();
  void kinect_get_state();
  void kinect_set_active_area(int xlow, int xhigh, int ylow, int yhigh, int zlow, int
      zhigh);
  void getMovement();
  void getBlobs();
  void emptyarray();

#endif
```

A.14 libfreenect.h

Code of libfreenect.h

```
/*
 * This file is part of the OpenKinect Project. http://www.openkinect.org
 *
 * Copyright (c) 2010 individual OpenKinect contributors. See the CONTRIB file
 * for details.
 *
 * This code is licensed to you under the terms of the Apache License, version
 * 2.0, or, at your option, the terms of the GNU General Public License,
 * version 2.0. See the APACHE20 and GPL2 files for the text of the licenses,
 * or the following URLs:
 * http://www.apache.org/licenses/LICENSE-2.0
 * http://www.gnu.org/licenses/gpl-2.0.txt
 *
 * If you redistribute this file in source form, modified or unmodified, you
 * may:
 *   1) Leave this header intact and distribute it under the same terms,
 *      accompanying it with the APACHE20 and GPL20 files, or
 *   2) Delete the Apache 2.0 clause and accompany it with the GPL2 file, or
 *   3) Delete the GPL v2 clause and accompany it with the APACHE20 file
 * In all cases you must keep the copyright notice intact and include a copy
 * of the CONTRIB file.
 *
 * Binary distributions must follow the binary distribution requirements of
 * either License.
 */

#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>
```

```
  typedef libusb_context freenect_usb_context; /**< Holds libusb-1.0 specific information */
#endif
//

/// If Win32, export all functions for DLL usage
#ifndef _WIN32
  #define FREENECTAPI /**< DLLExport information for windows, set to nothing on other platforms */
#else
  /**< DLLExport information for windows, set to nothing on other platforms */
  #ifdef __cplusplus
    #define FREENECTAPI extern "C" __declspec(dllexport)
  #else
    // this is required when building from a Win32 port of gcc without being
    // forced to compile all of the library files (.c) with g++...
    #define FREENECTAPI __declspec(dllexport)
  #endif
#endif

/// Enumeration of message logging levels
typedef enum {
 FREENECT_LOG_FATAL = 0,      /**< Log for crashing/non-recoverable errors */
 FREENECT_LOG_ERROR,          /**< Log for major errors */
 FREENECT_LOG_WARNING,        /**< Log for warning messages */
 FREENECT_LOG_NOTICE,         /**< Log for important messages */
 FREENECT_LOG_INFO,           /**< Log for normal messages */
 FREENECT_LOG_DEBUG,          /**< Log for useful development messages */
 FREENECT_LOG_SPEW,           /**< Log for slightly less useful messages */
 FREENECT_LOG_FLOOD,          /**< Log EVERYTHING. May slow performance. */
} freenect_loglevel;



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

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

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

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

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

/**
 * 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
```

```
/**
 * Retrieve the tilt state from a device
 *
 * @param dev Device to retrieve tilt state from
 *
 * @return The tilt state struct of the device
 */
FREENECTAPI freenect_raw_tilt_state* freenect_get_tilt_state(freenect_device *dev);

/**
 * Return the tilt state, in degrees with respect to the horizon
 *
 * @param state The tilt state struct from a device
 *
 * @return Current degree of tilt of the device
 */
FREENECTAPI double freenect_get_tilt_degs(freenect_raw_tilt_state *state);

/**
 * Set the tilt state of the device, in degrees with respect to the
 * horizon. Uses blocking control message call to update
 * device. Function return does not reflect state of device, device
 * may still be moving to new position after the function returns. Use
 * freenect_get_tilt_status() to find current movement state.
 *
 * @param dev Device to set tilt state
 * @param angle Angle the device should tilt to
 *
 * @return 0 on success, < 0 on error.
 */
FREENECTAPI int freenect_set_tilt_degs(freenect_device *dev, double angle);

/**
 * Return the movement state of the tilt motor (moving, stopped, etc...)
 *
 * @param state Raw state struct to get the tilt status code from
 *
 * @return Status code of the tilt device. See
 * freenect_tilt_status_code enum for more info.
 */
FREENECTAPI freenect_tilt_status_code freenect_get_tilt_status(freenect_raw_tilt_state *
    state);

/**
 * Set the state of the LED. Uses blocking control message call to
 * update device.
 *
 * @param dev Device to set the LED state
 * @param option LED state to set on device. See freenect_led_options enum.
 *
 * @return 0 on success, < 0 on error
 */
FREENECTAPI int freenect_set_led(freenect_device *dev, freenect_led_options option);

/**
 * Get the axis-based gravity adjusted accelerometer state, as laid
 * out via the accelerometer data sheet, which is available at
 *
 * http://www.kionix.com/Product%20Sheets/KXSD9%20Product%20Brief.pdf
 *
 * @param state State to extract accelerometer data from
 * @param x Stores X-axis accelerometer state
 * @param y Stores Y-axis accelerometer state
 * @param z Stores Z-axis accelerometer state
 */
FREENECTAPI void freenect_get_mks_accel(freenect_raw_tilt_state *state, double* x,
    double* y, double* z);

/**
 * Get the number of video camera modes supported by the driver.  This includes both RGB
      and IR modes.
 *
 * @return Number of video modes supported by the driver
 */
FREENECTAPI int freenect_get_video_mode_count();

/**
```

```
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  */
```

```
FREENECTAPI const freenect_frame_mode freenect_get_current_depth_mode(freenect_device *
    dev);

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

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


freenect_context *f_ctx;
freenect_device *f_dev;
freenect_video_format requested_format;
freenect_video_format current_format;


#ifdef __cplusplus
}
#endif

#endif //
```

A.15 threadfunc.h

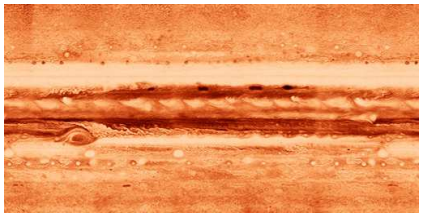
Code of threadfunc.h

```
#ifndef THREADFUNC_H
#define THREADFUNC_H
#include "libfreenect.h"

extern pthread_mutex_t gl_backbuf_mutex;
extern pthread_cond_t gl_frame_cond;
extern int got_rgb;
extern int got_depth;


//function prototypes
void *gl_threadfunc(void *arg);
//void depth_cb(freenect_device *dev, void *v_depth, uint32_t timestamp);
//rgb_cb(freenect_device *dev, void *rgb, uint32_t timestamp);
void *freenect_threadfunc(void *arg);

#endif
```

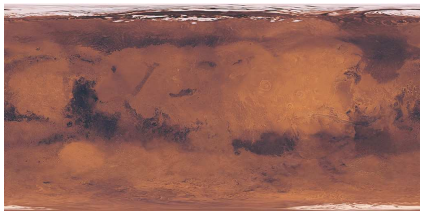
B Appendix B

B.1 textures

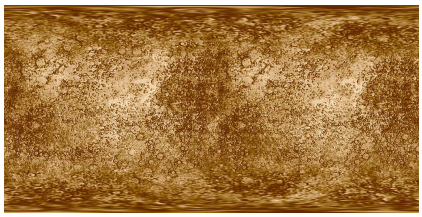
(a) Earth



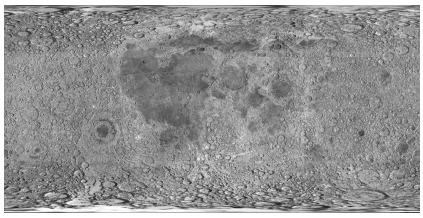
(b) Jupiter



(c) Mars



(d) Mercury



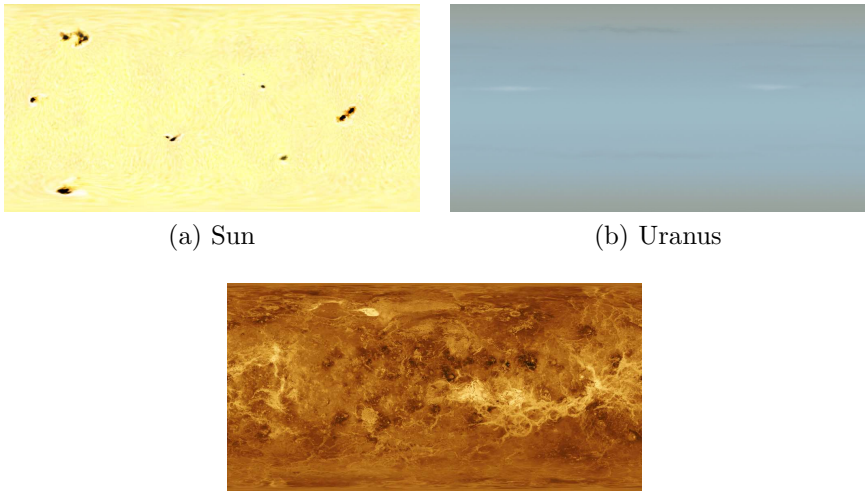
(e) Moon

(f) Neptune

(g) Pluto

(h) Saturn

Figure 11: Textures



(a) Sun (b) Uranus

(c) Venus

Figure 12: Textures