Penrose World

W4840 Final Project Report

Cong ZHU Yuan HUI Yao LUO

2014/5/14
Content

Overview .................................................................................................................................... 3
Hardware Organization ............................................................................................................. 4
  FPGA ......................................................................................................................................... 4
  Gamepad ................................................................................................................................. 4
  VGA Display ............................................................................................................................ 5
RayCasting ................................................................................................................................. 6
Image Generation ....................................................................................................................... 6
Object Rotation ......................................................................................................................... 7
Flow Path of the Entire Algorithm: .......................................................................................... 8
Visual Illusion Algorithm ........................................................................................................... 9
Function Description .................................................................................................................. 9
Pass Determination ................................................................................................................... 9
Three exceptions: ....................................................................................................................... 10
Movement of Ball ....................................................................................................................... 11
The movement of ball obeys the rules below: ......................................................................... 12
Real time dynamic 3D graphics calculation ............................................................................. 14
Acceleration by using parallel computing .............................................................................. 14
Tradeoff between resolution and refresh rate .......................................................................... 15
Separate one moving cube into four parts .............................................................................. 16
Lessons We Learnt ................................................................................................................... 17
Experience ................................................................................................................................. 18
Future Work ............................................................................................................................... 19
Contribution ............................................................................................................................... 20
Reference .................................................................................................................................... 21
Source Code .............................................................................................................................. 22
Software ..................................................................................................................................... 22
  Hello.c ...................................................................................................................................... 22
  Vga_led.c ................................................................................................................................ 55
  Vga_led.h ................................................................................................................................. 60
Makefile.............................................................................................................. 60
Hardware............................................................................................................... 61
Mifflegen.m......................................................................................................... 61
Sin_Cos_LUT.m..................................................................................................... 62
Vga_led.sv........................................................................................................... 62
Overview

In this project, we plan to implement a 3D game, in which, a diamond can move in an incredible way by utilizing the 3D space illusion feeling. The display of 3-D image involves large amount of point-to-point projection computation which possibly have no data correlation at all. Therefore it is very convenient to implement it by high speed parallel computation. Since the nature of software-based implementation is sequential execution, it is better to do these 3-D computation using hardware instead of software. With the FPGA source on this Sockit board, we can build such parallel computation hardware architecture easily and therefore increase the processing speed of 3-D image. In this game, players can change the angle of view to create a route, which doesn’t exist in 3D world. In particular, it is based on the concept of “Penrose Stair”. (“Penrose Stair” is a two-dimensional depiction of a staircase in which the stairs make four 90-degree turns as they ascend or descend yet form a continuous loop, so that a person could climb them forever and never get any higher.)

This project is similar to a PSP game “Echochrome”. A video of that can be found on YouTube: http://www.youtube.com/watch?v=QfICeBtvv8U
Hardware Organization

The components of our system include the HPS and FPGA core on the Sockit Board. One task for the FPGA is mainly used to build the controller (driver) of VGA and the gamepad. The FPGA is also responsible for the parallel computation of point projection on screen, generate input data stream into the VGA screen. We use gamepad to control direction of rotation, store the location information of 3-D object and its 2-D projection in the on-chip RAM and store the trigonometric value of rotation angle in the ROM. Finally, all the communications are happened on the Avlon bus. Further details about each part is shown as bellow:

**FPGA**

Besides the controller of all the peripherals, FPGA is also responsible for the parallel computation of each point on the object onto the screen. In each clock cycle, we generate several rays which pass through the eye and each point on the surface of target object, record the two dimensional location on the screen and calculate the distance from the screen to the object’s surface. We can do the calculation of each line in parallel for the reason that every single computation has no data dependency with others. The detail of every calculation is shown in the algorithm implementation.

**Gamepad**

We use the gamepad to control the rotation of 3D object showing on the screen as well as the game control logic. There are four basic moving instructions in our design which controls the direction of rotation. Beyond that, several key combinations are also applied to the game logic so that we can have multiple rotation speed and can change cube location timely. The press of button will be detected by the gamepad and send to the HPS as an internal signal. Gamepad communicate with the main logic in HPS using USB port and Avlon bus. These
moving instructions then define the moving operations and the angle of moving to the FPGA to generate the new rotated 3-D object. The gamepad is connected to the Sockit board using USB ports on the board. The USB is controlled by SMSC USB3300 controller.

VGA Display

The data processed by main logic in FPGA and HPS has the form of a 3-D array in which the first two columns are the X-Y coordinates of points that will display on the VGA screen. The third column indicates which surface the current shape is projected from. The VGA controller will first analyze the coordinates of input data to decide that which pixel on the screen is chosen to be light up. Then, the third column of input data will be used to decide the color of this projected surface in order to distinguish with different surfaces. Location of the cubes are given by the HPS so that we can change the location arbitrarily. The whole protocol is much like the lab3 where VGA controller response only for lighting up specific area of the screen based on the input data. More details will be given in the following chapters.
RayCasting

Image Generation

In our design we use an algorithm called Ray-Casting to implement 3-D projection of objects onto the VGA screen. Traditional method of Ray-Casting sets a fixed point called ‘eye point’ at one side of the screen and the object being projecting is located at the other side of the screen. The screen is called image plane since the object will be projected onto it and form a 2-D image on the image plane. For the convenience of our discussion, the object is divided into several points. We form a ray starting from the eye point, passing through the image plane and extending further on its direction. If this ray ‘hits’ at one of the points on the object, i.e. one of the points on the object falls onto the ray, than this specific point of the object should be light up on the image plane. The location of this light up point would be the point on the image plane that lays on the ray. If multiple points of the object hit the ray, we simply consider the first point, i.e. the one with minimal distance to the eye point, and ignore the other options so that light up points on the image plane will always correspond to the nearest point on the object. The schematic plot is shown as below:

Figure 1

The main advantage of this method is that ray casting can show the shielding between multiple objects as well as distortion introduced by the difference of distance at different points on the object. This property assures that we can capture the right location relationship between multiple objects on the image plane. However, in our project, we intend to create an image of target objects on the screen that actually ignore the distortion caused by the difference in distance. Only under this particular scenario, a visual illusion will occur that forms a path between two objects that should never been formed in the real world. To achieve this, we apply a litter change to the original algorithm of the Ray-Casting. Instead of fixing a single eye pint where all the rays start at, we do a parallel projection of the objects onto the image plane. Each pixel on the image plane now have its own ray that targeting at a direction that is perpendicular to the image plane. The schematic plot of revised version of Ray-Casting is shown as below:
Object Rotation

Every point of the target object are described using bunch of points. Each of the points has a three dimensional coordinate that stores the location information. Therefore, the rotation of object is simply a matrix multiplication that map the original coordinate to a new location using rotation matrix. The rotation matrix is illustrated as bellow:

\[
\begin{align*}
\text{Rot}_X(\theta) &= 
\begin{bmatrix}
1 & 0 & 0 \\
0 & \cos\theta & -\sin\theta \\
0 & \sin\theta & \cos\theta
\end{bmatrix} \\
\text{Rot}_Y(\gamma) &= 
\begin{bmatrix}
\cos\gamma & 0 & \sin\gamma \\
0 & 1 & 0 \\
-\sin\gamma & 0 & \cos\gamma
\end{bmatrix} \\
\text{Rot}_Z(\varphi) &= 
\begin{bmatrix}
\cos\varphi & -\sin\varphi & 0 \\
\sin\varphi & \cos\varphi & 0 \\
0 & 0 & 1
\end{bmatrix}
\end{align*}
\]

Denote the original location of point as a three dimensional vector \( P \) and the mapped location as \( P' \):

\[
P = 
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}, \quad
P' = 
\begin{bmatrix}
x' \\
y' \\
z'
\end{bmatrix}
\]

The transformation equation is illustrated as bellow:
\[ P = \text{Rot}_X(\theta)\text{Rot}_Y(\gamma)\text{Rot}_Z(\phi)P' \]

Where \(\theta, \gamma, \phi\) is the corresponding angle of rotation with respect to the x-axis, y-axis and z-axis.

Flow Path of the Entire Algorithm:

1. Determine the boundary conditions of each cube. We consider only cube generations in our design. There are six surfaces in a cube. Each two of them are parallel to the other. Therefore for each cube, we should specify both the length, the width and the location of the surface. This constraint information also determines the relative location of all the objects, length and direction of each object.

2. Initialize and record the value of a three-dimension vector \( R(\theta, \gamma, \phi) \) which represents the rotation angles according to x-axis, y-axis and z-axis. After some mathematic operations describe later, we should be able to get the post-rotate location information of each object.

3. Set a reasonable step value to divide the boundary conditions into a boundary vector to make it more feasible to be applied with a numerical calculation. Also, set up a small noise margin of each point so that the ray wouldn’t be actually hit the point to be considered as a hit. This improves the algorithm’s robustness and make the image on the screen more stable.

4. Forming 256*256 pseudo pixel on the screen. Each of them consists 2*2=4 real pixels. Use these pseudo pixels as the starting point of each ray. Each clock cycle we push the rays a step further toward the object space. If any of the ray ends up at any of the object’s surface, light up the corresponding pseudo pixel area on the screen to reflect a hit on the object.

5. If any of the rays hit at the surface of object, check the number of object and surface. Records the number into the buffer and color the screen based on the pre-defined mapping relationship between these numbers and different colors. If multiple hit points are detected, use a flag to indicate which one is the nearest one to the image plane.
Visual Illusion Algorithm

Function Description

**Structure cp_node**: stores the information such as location, projection vector matrix to every cubic.

**Structure balls**: store all information related to ball, such as previous and present location, direction in order to decide the location of each ball in each time slot.

**Structure Plane_info**: store all information related to cubic, it will show us if the cubic is connect to other cubic and the connection point.

**initialize_ball()**: initialize all ball structure at the beginning of game.

**initialize_endPoint()**: calculate the endpoint at the beginning of game, which will related to the graph we create.

**initialize_plane()**: initialize all plane structure at the beginning of game.

**ball_moving()**: control the movement of all balls in each time slot.

**ball_changeplane(int ballnum)**: when balls jump from one cubic to the other cubic in suitable projection view, the information of ball change by this function.

**calculate_jump_point(int ball_num)**: when the direction of balls changed, this function is used to calculated if there are jump points which equal to the paths to other cubic exist.

**print_segment_info()**: Read and print the segment values

**write_segments(const unsigned char segs[64])**: Write the contents of the array to the display

**int main()**: the main function initialized the space at first, then listen to gamepads signal and call ball moving function.

**custom_design(struct usb_keyboard_packet packet, int transferred)**: enable players to generate the size and location of cubic by themselves.

**generate_cubic()**: generate cubic with given information.

**initialize()**: initialize all game information when apply a new map.

**calculate_pass()**: when projection view changing, calculate all the connection information between every two cubic.

**transfervalue(int op)**: change the projection view by the information passed from gamepad.

Pass Determination

We use a interesting method to implement pass determination which means determine if two cubic is connected in a given direction condition.

At first, we construct cubic by many smaller cubic with length, width and height equal to 10. For each cubic, it has two invisible small cubic in endpoint. Those invisible cubic is used to decided if we can think two cubic are connected or no. The stretch graph is shown as figure 3. For each invisible cubic, it has a center node c’. For each long cubic a finite length axle wire is composed by many nodes n’ with step length 1 extended in the long cubic direction. For all n’, if there exist a center node c’ in different cubic, the connected line between n’ and c’ is vertical the projection plane, the two cubic is connected by c’ and n’. We can see from figure 4, in view of the projection
direction, two cubic are connected in our angle of view.

Three exceptions:

In the implement of the algorithm above, we find there are also some bugs. Which mean in those case, it will satisfy the algorithm condition above, however, in the view of user, the projection result is not reasonable for cubic to be connected with each other.

To make up, we will add 2 more exception to avoid those unreasonable cases. We will list the cases and related solution method.

case1: For this case, we can add the condition that if the vector from center node c’ to n’ point out to screen. As we make our normal vector point into the screen, which means if two vector is opposite, we should not consider the two cubic are connected.
case2: if two cubic are parallel, we should not allow the center node c’ below connected n’, we implement this by the method motioned in case1 as well. Also, there is the other problem, if the marked location connected, player can’t see it, we should no allow this happen. So we implement this by check if the connected vectors direction is pointed into the screen, which mean parallel to the normal vector. We should no allow this to happen.

**Movement of Ball**

The operation of movement of ball is complex. In this section, we will discuss the information variables which is stored in structure balls and their usage.

<table>
<thead>
<tr>
<th>Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>plane_num</td>
<td>The plane number the ball is in</td>
</tr>
<tr>
<td>Bx</td>
<td>The x-coordinate of ball</td>
</tr>
<tr>
<td>By</td>
<td>The y-coordinate of ball</td>
</tr>
<tr>
<td>Bz</td>
<td>The z-coordinate of ball</td>
</tr>
<tr>
<td>b_dict</td>
<td>The moving direction of ball</td>
</tr>
<tr>
<td>next_jump_p</td>
<td>Whether there is jump point in front of the path</td>
</tr>
<tr>
<td>jump_x</td>
<td>The x-coordinate of jump point we the ball jump from</td>
</tr>
<tr>
<td>Jump_y</td>
<td>The y-coordinate of jump point we the ball jump from</td>
</tr>
<tr>
<td>Jump_z</td>
<td>The z-coordinate of jump point we the ball jump from</td>
</tr>
<tr>
<td>jump_Node</td>
<td>The serial number of the invisible the ball will jump to</td>
</tr>
<tr>
<td>prior_plane</td>
<td>the prior plane number of ball</td>
</tr>
<tr>
<td>prior_direct</td>
<td>the prior direction number of ball</td>
</tr>
<tr>
<td>next_is_edge</td>
<td>Whether the bound edge in forward path is connected to other cubic</td>
</tr>
<tr>
<td>edge_plane_num</td>
<td>If the bound edge in forward path is connected to other cubic, record the number of that cubic</td>
</tr>
<tr>
<td>ret_bx</td>
<td>If the ball is in protection state, it will record the x coordinate value for recover</td>
</tr>
</tbody>
</table>
**The movement of ball obeys the rules below:**

1. At the begin of the game, there are five cubic \{c1, c2, c3, c4, c5\}, the endpoint is located in c5, four balls \{b1, b2, b3, b4\} is placed in the start point of c1-c4. We initialize the location and ball.plane_num of balls which is based on the location of each plane. The ball.plane_num indicate which cubic the ball is in, and the change of balls location will be based on this variable, the ball will move toward the direction its located cubic extend. As we have define all start point of cubic is less than endpoint of cubic, the direction of ball are set to 1 initially which mean the balls location will increase at first.

2. Balls will keep moving in each time slot in a given direction until it reach edge or jump point.

3. Balls will calculate if there is jump points in front of them once the direction of it is set or changed or the rotation angle of projection screen change.

4. If there is no jump point in front of ball, then balls reaches the endpoint of the cubic it is in, then it change its direction to remove backward.

5. If there is a jump point in front of balls. There are two case we should discuss. (a). the ball moves from edge of a cubic to a node on axle of the other cubic. Just shown in figure7, (b). the ball moves from a node on axle of cubic to the edge of the other cubic, just shown in figure8.

   In case a, when the ball moves to the edge of cubic A, it will move 5 more step, and then change its coordinate to the coordinate of center node c’. During this process, it is necessary to have a recover protection. Because when rotation angle change, as the ball is not in both two cubic, it may deorbit in this case. So if the ball is in recover protection, when rotation angle changes, we will put the ball in the jump point of this previous cubic to avoid this bug.

   In case b, it is simpler. The when the ball reach the jump point, it will change its location and to the connected center node c’, and change its other related information as well. We have design a method to let the ball run into the right orbit after jumping.

**Figure 7**
The flow chart for the ball moving control is shown below.

Figure 8

Figure 9
Hardware Implementation Details

Real time dynamic 3D graphics calculation

Hardware implementation is the core part of our project. In our implementation, we do ray casting through every pixel and dynamically configure the image which shown on the screen. To generate image on VGA screen, we have to finish two steps: rotating the object and ray casting. One major disadvantage of this method is that, before ray casting, we have to generate the rotated object and store it into a buffer. Which consume time and memory resource. To solve this problem, we used a trick method to merge these two steps into one step. We do ray casting for every pixel and rotate the casting ray in the opposite direction at the sometime. Thus, the rotation can be done at the same time as ray casting, which accelerated the calculation and avoided the necessity of storing rotated object.

Besides, to achieve real time calculation, we have to do ray casting while showing the image. We divided the screen into two parts, odd lines and even lines. For each part, we built a frame buffer. When scanning the odd line, VGA read from the odd line frame buffer to show the image on the screen and write the calculation result into the even line frame buffer. When scanning the even line, VGA read from the even line frame buffer to show the image on the screen and write the calculation result into the even line frame buffer.

Acceleration by using parallel computing

If the resolution of screen is N*N, we have to do ray casting in an N*N*N space. Since the casting ray can only move one step further per clock cycle, if we can calculate those pixels one by one, it will take N*N*N cycles to update the value in frame buffer. That will make VGA refresh rate extremely slow, as the result, objects on screen cannot move smoothly.

To solve this problem, we used parallel computing to accelerate the ray casting calculation. In our project, the VGA resolution is 256*256. Actually, we merged 4 pixels into a single one to achieve this lower resolution. I will explain late why using this lower resolution. As shown in figure 3, we group 16 pixels together. In each iteration, we calculate 16 pixels in parallel. Each iteration took 256 clock cycles. For each row of pixels, we have 16 such groups. Since we used a lower resolution, we only need to calculate a single row while scanning two lines on the screen, which contains 3200 clock cycles. It is insufficient to calculate all pixels of a single row. Thus, we separated the each row into two parts: the left part and the right part and alternatively calculates these two parts.
Tradeoff between resolution and refresh rate

We chose a lower resolution because of the tradeoff between resolution and refresh rate. Refresh
rate equals to the speed we update frame buffer value. Since the pixel group size cannot be arbitrary larger (16 is the largest group size without violating timing constrain), and we cannot change the screen scan rate, lower the resolution is a good way to achieve high refresh rate.

If we change the resolution from $N \times N$ into $(N/2) \times (N/2)$, the number of pixels need to be calculated will be $1/4$ of the original one. Meantime, the ray casting depth will be $1/2$ of the original one because we cast in a cubic space. As a result refresh rate will be 8 times of the original one.

If we use the resolution of $640 \times 480$, it will take 269ms to refresh the image. It is too slow that, the object cannot move smoothly. When we change the resolution into $256 \times 256$. It only took 34ms to refresh the image.

**Separate one moving cube into four parts**

When the game start, we separate an integrated moving cube into four parts and can merge them back into an integrated one during the game (as shown in figure 4). Actually, it is a visual illusion. We created four integrated moving cubes but just show 1/4 surface of each one. Different cubes show different part of their surface. When merging them into an integrated one. The whole surface will be shown. We assigned different color to these four cubes. Thus, the integrated one looks like a magic cube. We achieved it by alternatively highlight different parts of surface of each cube by using a MUX and changing its select single every clock cycle.
Lessons We Learnt

The first thing we learnt during this project is that how to communicate with others effectively when all the group numbers are doing their works separately. Leave comments on all the variable declarations so that others can understand what you are saying. The other thing is that divide all the works done by different people into different parts. This is particularly useful in hardware codes since the order of the code doesn’t affect its functionality at most of the time. By doing this we could understand other’s work much easier.

The other thing that we learnt is always to remember to consider how many resources we can actually use on the Board. The complexity of our project is much higher than that of a simple function block or module. When doing the ray-casting, if we calculate all the pixels on screen at one clock cycle, it would never work because this FPGA device cannot actually handle this scale of data at one time based on the number of LUT units and DSP blocks it have. Besides, matrix multiplication involves using bunch of DSP blocks. If multiple multiplications are cascading together, the critical path delay may exceed the timing constraint settled by the board’s PLL generated clock. This may result in a lot of noise points on the screen. To archive timing closure as well as functionality robustness, we have to use and organize resources on board with careful considerations. For a VGA screen with refresh rate at 25Mhz, there is no need to calculate all the pixel point at one cycle since the user cannot see the difference if we use a lower computation frequency. Therefore, we divide the computation of the whole pixel into several parts and handle them separately. By doing this we actually reduce the usage of logic units from almost 100% to less than 50% and DSP blocks from 103% to 63% while eliminate all the noise points generated by timing violations.

The last thing we learnt from this project is to let the hardware and software do the right works. This means that we should let the hardware do the 3-D projection and shielding while let the software control the game logic and visual illusion. This allocation is based on the fact that image processing algorithms such as Ray-Casting could be done more efficient in parallel while the game logic and visual illusion has a nature of consequential. We actually implement game control in the hardware. It is much more painful and complex compare the later version implemented in software for the reason that hardware compiling is much more time consuming and hard to describe.
Experience

After one semester’s learning of SoCkit Board and the usage of Quartus II design suit, we are familiarized with not only HDL languages such as SystemVerilog and Verilog, but also learnt about hardware architecture of cyclone V and HPS cores. At first we spent quite a lot of time in dealing with signed values using Verilog. Verilog is not very suitable for handling signed values such as integer. The transformation between logic and integer value should be taken extra care of because it can easily overflow during transition.

Besides, some useful tools are also introduced in lecture and project such as System Console, Qsys, SignalTap and MegaWizard. For testing of peripherals, SignalTap is always a powerful tool because it can snoop the real time signal status in FPGA. By using SignalTap we can now actually see how the signals are effected in devices such as DDR3 SDRAM which has a relative complex timing diagram and control logic. And the Qsys as well as Megawizard can provide simple and effective solutions to deal with interconnecting between IP cores and private modules.

Finally, to implement the RayCasting algorithm and visual illusion is also a very challenging work. The math behind object projection and shielding can be very abstract, which makes debug in software as well as hardware difficult.
Future Work

1. The resolution we select to display object on screen is relative low due to limit resources in our chip. In our hardware design, we use several DSP blocks to implement the multiply of the rotation matrix. Cascading of these operations increase the path delay in our circuit significantly so that we have to separate the display operation into multi-cycles. This trade off decrease the refresh rate of VGA screen but decrease the timing violation as well. Therefore the next thing we should do would be optimize our logical path so that resources on chip can be allocated with more efficiency. Furthermore, pipeline architecture could be introduced into our design so that we can achieve higher throughput and reduce critical path delay.

2. Our design didn’t use the on board DDR3 SDRAM for the reason that RAM and ROM resource are enough for image computing. But we can still use the DDR3 SDRAM for other purpose such as background music playing.

3. Due to the limitation of computing resource (LUT), we choose to decrease pixel resolution to accommodate the large parallel computing requirement of RayCasting. There are some other methods that can acquire same performance compare to RayCasting such as Z-Buffering. We can explore other solutions and get same performance as well.
**Contribution**

Yuanhui LUO (yl3026): Software, Game Logic design, Gamepad controller design;
Cong ZHU (cz2311): Algorithm design, Hardware design;
Yao LUO: Algorithm design, Hardware design;
Reference

[4] ‘Volume Visualization With Ray Casting’
[5] ‘Volume Rendering’
Source Code

Software

Hello.c

/*
 * Userspace program that communicates with the led_vga device driver
 * primarily through ioctls
 *
 * Stephen A. Edwards
 * Revised by Penrose
 *
 * Columbia University
 */

#include <stdio.h>
#include "vga_led.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include "usbkeyboard.h"
#include<math.h>

#define BUFFER_size 128
#define cubnum 5

struct cpnode{
    int plane_num;
    float x_node, y_node, z_node;
    float x_va[256];
    float y_va[256];
    float z_va[256];
}
struct Ball{
    int plane_num;      // the plane the Ball stays in
    int bx, by, bz;
    int b_dict;
    int next_jump_p;
    int jump_x, jump_y, jump_z;
    int jump_Node;
    int prior_plane;
    int prior_direct;
    int next_is_edge;
    int edge_plane_num;
    int outedge;
    int ret_bx, ret_by, ret_bz;
    int ret_direct;
    int return_protect;
}balls[cubnum-1];

struct Plane{
    int connectpoint_num;
    int N_connect[cubnum-1];
    int P_x[cubnum-1];
    int P_y[cubnum-1];
    int P_z[cubnum-1];
    int edge_node[2];
    int edge_node_connect[2];
    int edge_node_jumpx[2];
    int edge_node_jumpy[2];
    int edge_node_jumpz[2];
    int edge_node_jump_plane[2];
}Plane_info[cubnum];

float D = 0;
float D2 = 0, D3;
float precision = 0.01;
int cubic[8];
float v_x,v_y,v_z;
float v_x1,v_y1,v_z1;
float v_x2,v_y2,v_z2;
double rotate_angle_x, rotate_angle_y;
int cubic_nodes_num[cubnum];
int record_location1x, record_location2x, record_location1y, record_location2y, record_location1z, record_location2z;
int seeSurface1, nonSeeSurface2, parallel;

struct libusb_device_handle *keyboard;
uint8_t endpoint_address;
int sub0;
int sub1;
int sub2;
int vga_led_fd;
int x_angle_l;
int y_angle_l;
int x_angle_h;
int y_angle_h;
int rotation_rate = 0;
int flag = 0;
int ball_reach[cubnum - 1];

// end point info
int x_ep;
int y_ep;
int z_ep;

// trans cubics info
int extend_d[cubnum];
int x_s[cubnum];
int x_e[cubnum];
int y_s[cubnum];
int y_e[cubnum];
int z_s[cubnum];
int z_e[cubnum];

int extend_d1[] = {3, 1, 3, 1, 1};
int x_s1[] = {-50, -40, 40, -20, -50};
int x_e1[] = {-40, 20, 50, 50, 20};
int y_s1[] = {-50, 40, 0, -20, -20};
int y_e1[] = {-40, 50, 10, -10, -10};
int z_s1[] = {-30, -50, -10, 40, 0};
int z_e1[] = {50, -40, 30, 50, 10};

int extend_d2[] = {3, 3, 3, 3, 3};
int x_s2[] = {-50, -20, 10, 40, 40};
int x_e2[] = {-40, -10, 20, 50, 50};
int y_s2[] = {-50, 40, -50, 40, -50};
int y_e2[] = {-40, 50, -40, 50, -40};
int z_s2[] = {10, -50, -50, -50, -10};
int z_e2[] = {50, 10, 10, 10, 50};

int extend_d3[] = {3,1,3,1,1};
int x_s3[] = {-50, -40, 40, -40, -5};
int x_e3[] = {-40, 40, 50, 40, 25};
int y_s3[] = {30, 30, 30, 30, -5};
int y_e3[] = {40, 40, 40, 40, 5};
int z_s3[] = {-40, -50, -40, 40, -5};
int z_e3[] = {40, -40, 40, 50, 5};

int extend_d4[] = {3,1,3,1,3};
int x_s4[] = {-50, -50, 10, -10, 40};
int x_e4[] = {-40, 10, 20, 50, 50};
int y_s4[] = {-50, 40, -50, 40, -50};
int y_e4[] = {-40, 50, -50, 50, -40};
int z_s4[] = {-10, -40, -50, 30, -10};
int z_e4[] = {50, -30, 10, 40, 50};

int extend_d5[] = {3,1,3,1,3};
int x_s5[] = {-50, -30, 20, 20, 0};
int x_e5[] = {-40, 30, 30, 80, 10};
int y_s5[] = {-10, 40, 40, -20, -50};
int y_e5[] = {0, 50, 50, -10, -40};
int z_s5[] = {-50, 40, -30, 40, -10};
int z_e5[] = {10, 50, 30, 50, 50};

int extend_d6[] = {1,1,1,1,1};
int x_s6[] = {-5, -5, -5, -5, -5};
int x_e6[] = {5, 5, 5, 5, 5};
int y_s6[] = {-5, -5, -5, -5, -5};
int y_e6[] = {5, 5, 5, 5, 5};
int z_s6[] = {-5, -5, -5, -5, -5};
int z_e6[] = {5, 5, 5, 5, 5};

int width = 5;

static unsigned char message[VGA_LED_DIGITS] =
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
//ball control
void initialize_ball(){
    int bii,wi;
    for(bii=0;bii<4;bii++){
        balls[bii].plane_num = bii;
        balls[bii].bx = x_s[bii] + 5;
        balls[bii].by = y_s[bii] + 5;
        balls[bii].bz = z_s[bii] + 5;
        balls[bii].b_dict = 1;
        balls[bii].next_jump_p = 0;
        balls[bii].prior_plane = -1;
        balls[bii].prior_direct = 1;
        balls[bii].next_is_edge = 0;
        balls[bii].outedge = 0;
    }

    for(wi = 0;wi<4;wi++){
        message[40+wi*3] = balls[wi].bx;
        message[41+wi*3] = balls[wi].by-10;
        message[42+wi*3] = balls[wi].bz;
    }
    write_segments(message);
}

void initialize_endPoint(){
    x_ep = (x_s[4] + x_e[4])/2;
    y_ep = y_s[4];
    z_ep = (z_s[4] + z_e[4])/2;

    message[7] = x_ep;
    message[8] = y_ep;
    message[9] = z_ep;
}

void initialize_plane(){
    int i;
    for(i = 0;i<cubnum;i++){
        Plane_info[i].connectpoint_num = 0;
        Plane_info[i].edge_node_connect[0] = 0;
        Plane_info[i].edge_node_connect[1] = 0;
    }
void ball_moving(){

    int bi, wi;
    int kk, kk2;
    for (bi = 0; bi < cubnum - 1; bi++) {
        if (balls[bi].bx == x_ep && balls[bi].by == (y_ep + 5) &
            & balls[bi].bz == z_ep) ball_reach[bi] = 1;
        if (ball_reach[bi] == 1) continue;

        if ((balls[bi].bx >= (x_s[balls[bi].plane_num] + 5) &
                & balls[bi].bx <= (x_e[balls[bi].plane_num] -
            5) & & balls[bi].by >= (y_s[balls[bi].plane_num] + 5) & &
                & balls[bi].by <= (y_e[balls[bi].plane_num] -
            5) & & balls[bi].bz >= (z_s[balls[bi].plane_num] + 5) & & balls[bi].bz <= (z_e[balls[bi].plane_num] -
            5)) || (balls[bi].next_is_edge == 1) {
            if (balls[bi].next_jump_p == 0) {
                switch (extend_d[balls[bi].plane_num]) {
                    case 1:
                        if (balls[bi].b_dict == 1) {
                            if (balls[bi].bx < x_e[balls[bi].plane_num] - 5) {
                                balls[bi].bx = balls[bi].bx + balls[bi].b_dict;
                            }
                        }
                        else {
                            balls[bi].b_dict = -balls[bi].b_dict;
                        }
                calculate_jump_point(bi);
            }
            else if (balls[bi].b_dict == -1) {
                if (balls[bi].bx > x_s[balls[bi].plane_num] + 5) {
                    balls[bi].bx = balls[bi].bx + balls[bi].b_dict;
                }
                else {
                    balls[bi].b_dict = -balls[bi].b_dict;
                }
            }
            break;
        }
        case 2:
        }
    }
}

int bi, wi;
int kk, kk2;
for (bi = 0; bi < cubnum - 1; bi++) {
if (balls[bi].bx == x_ep & & balls[bi].by == (y_ep + 5) & &
    balls[bi].bz == z_ep) ball_reach[bi] = 1;
if (ball_reach[bi] == 1) continue;

if ((balls[bi].bx >= (x_s[balls[bi].plane_num] + 5) & &
        balls[bi].bx <= (x_e[balls[bi].plane_num] -
    5) & & balls[bi].by >= (y_s[balls[bi].plane_num] + 5) & &
        balls[bi].by <= (y_e[balls[bi].plane_num] -
    5) & & balls[bi].bz >= (z_s[balls[bi].plane_num] + 5) & &
        balls[bi].bz <= (z_e[balls[bi].plane_num] -
    5)) || (balls[bi].next_is_edge == 1){
if (balls[bi].next_jump_p == 0) {
    switch (extend_d[balls[bi].plane_num]) {
        case 1:
            if (balls[bi].b_dict == 1) {
                if (balls[bi].bx < x_e[balls[bi].plane_num] - 5) {
                    balls[bi].bx = balls[bi].bx + balls[bi].b_dict;
                }
            }
            else {
                balls[bi].b_dict = -balls[bi].b_dict;
            }
        calculate_jump_point(bi);
    }
    else if (balls[bi].b_dict == -1) {
        if (balls[bi].bx > x_s[balls[bi].plane_num] + 5) {
            balls[bi].bx = balls[bi].bx + balls[bi].b_dict;
        }
        else {
            balls[bi].b_dict = -balls[bi].b_dict;
        }
    }
    break;
}
else{
    balls[bi].b_dict = -balls[bi].b_dict;
    calculate_jump_point(bi);
}
}

} else if(balls[bi].b_dict == -1){
    if (balls[bi].by > \(y_S[balls[bi].plane_num] + 5\) ) {
        balls[bi].by = balls[bi].by + balls[bi].b_dict;
    }
    else{
        balls[bi].b_dict = -balls[bi].b_dict;
        calculate_jump_point(bi);
    }
}
break;

} case 3:{
    if (balls[bi].b_dict == 1) {
        if (balls[bi].bz < \(z_E[balls[bi].plane_num] - 5\) ) {
            balls[bi].bz = balls[bi].bz + balls[bi].b_dict;
        }
        else{
            balls[bi].b_dict = -balls[bi].b_dict;
            calculate_jump_point(bi);
        }
    }
    else if(balls[bi].b_dict == -1){
        if (balls[bi].bz > \(z_S[balls[bi].plane_num] + 5\) ) {
            balls[bi].bz = balls[bi].bz + balls[bi].b_dict;
        }
        else{
            balls[bi].b_dict = -balls[bi].b_dict;
            calculate_jump_point(bi);
        }
    }
    break;
}

}

default:
    break;
}
else if(balls[bi].next_jump_p == 1&&balls[bi].next_is_edge == 0){
    switch (extend_d[balls[bi].plane_num]) {
    case 1:
        if (balls[bi].b_dict == 1) {
            if (balls[bi].bx < balls[bi].jump_x) {
                balls[bi].bx = balls[bi].bx + balls[bi].b_dict;
            } else {
                ball_changeplane(bi);
                calculate_jump_point(bi);
            }
        } else if (balls[bi].b_dict == -1) {
            if (balls[bi].bx > balls[bi].jump_x) {
                balls[bi].bx = balls[bi].bx + balls[bi].b_dict;
            } else {
                ball_changeplane(bi);
                calculate_jump_point(bi);
            }
        }
        break;
    }
    case 2:
    } else if (balls[bi].b_dict == 1) {
        if (balls[bi].by < balls[bi].jump_y) {
            balls[bi].by = balls[bi].by + balls[bi].b_dict;
        } else {
            ball_changeplane(bi);
            calculate_jump_point(bi);
        }
    } else if (balls[bi].b_dict == -1) {
        if (balls[bi].by > balls[bi].jump_y) {
            balls[bi].by = balls[bi].by + balls[bi].b_dict;
        } else {
            ball_changeplane(bi);
            calculate_jump_point(bi);
        }
    } break;}
case 3:
    if (balls[bi].b_dict == 1) {
        if (balls[bi].bz < balls[bi].jump_z) {
            balls[bi].bz = balls[bi].bz + balls[bi].b_dict;
        }
        else{
            ball_changeplane(bi);
            calculate_jump_point(bi);
        }
    }
    else if(balls[bi].b_dict == -1){
        if (balls[bi].bz > balls[bi].jump_z) {
            balls[bi].bz = balls[bi].bz + balls[bi].b_dict;
        }
        else{
            ball_changeplane(bi);
            calculate_jump_point(bi);
        }
    }
    break;
}
else if(balls[bi].next_jump_p == 1&&balls[bi].next_is_edge == 1){
    balls[bi].ret_bx = balls[bi].jump_x;
    balls[bi].ret_by = balls[bi].jump_y;
    balls[bi].ret_bz = balls[bi].jump_z;
    balls[bi].ret_direct = balls[bi].plane_num;

    switch (extend_d[balls[bi].plane_num]) {
    case 1:
        if (balls[bi].b_dict == 1) {
            if (balls[bi].bx >= x_e[balls[bi].plane_num] - 5)balls[bi].return_protect = 1;
else balls[bi].return_protect = 0;
    if (balls[bi].bx < x_e[balls[bi].plane_num] + 5) {
        balls[bi].bx = balls[bi].bx + balls[bi].b_dict;
    } else {
        ball_changeplane(bi);
        calculate_jump_point(bi);
    } }
    else if (balls[bi].b_dict == -1) {
        if (balls[bi].bx <= x_s[balls[bi].plane_num] + 5) balls[bi].return_protect = 1;
        else balls[bi].return_protect = 0;
        if (balls[bi].bx > x_s[balls[bi].plane_num] - 5) {
            balls[bi].bx = balls[bi].bx + balls[bi].b_dict;
        } else {
            ball_changeplane(bi);
            calculate_jump_point(bi);
        }
        break;
    }
    case 2: {
        if (balls[bi].b_dict == 1) {
            if (balls[bi].by >= y_e[balls[bi].plane_num] - 5) balls[bi].return_protect = 1;
            else balls[bi].return_protect = 0;
            if (balls[bi].by < y_e[balls[bi].plane_num] + 5) {
                balls[bi].by = balls[bi].by + balls[bi].b_dict;
            } else {
                ball_changeplane(bi);
                calculate_jump_point(bi);
            }
        } else if (balls[bi].b_dict == -1) {
            if (balls[bi].by <= y_s[balls[bi].plane_num] + 5) balls[bi].return_protect = 1;
            else balls[bi].return_protect = 0;
            if (balls[bi].by > y_s[balls[bi].plane_num] - 5) {
                balls[bi].by = balls[bi].by + balls[bi].b_dict;
            } else {
                ball_changeplane(bi);
                calculate_jump_point(bi);
            }
        } else
    }
```c
//
//

break;
}

switch (balls[bi].b_dict)
{
    case 3:
        if (balls[bi].b_dict == 1) {
            if (balls[bi].bz >= z_e[balls[bi].plane_num] - 5) balls[bi].return_protect = 1;
            else balls[bi].return_protect = 0;
            if (balls[bi].bz < z_e[balls[bi].plane_num] - 5) {
                balls[bi].bz = balls[bi].bz + balls[bi].b_dict;
            }
        }
        else if(balls[bi].b_dict == -1){
            if (balls[bi].bz <= z_s[balls[bi].plane_num] + 5) balls[bi].return_protect = 1;
            else balls[bi].return_protect = 0;
            if (balls[bi].bz > z_s[balls[bi].plane_num] - 5) {
                balls[bi].bz = balls[bi].bz + balls[bi].b_dict;
            }
        }
        ball_changeplane(bi);
        calculate_jump_point(bi);
    }
    break;
}

default:
    break;
}
```

```c
else{
    //printf("try");
    switch (extend_d[balls[bi].plane_num]){
        case 1: {
            if(balls[bi].b_dict==1&&balls[bi].bx<x_s[balls[bi].plane_num]-5)balls[bi].b_dict=1;
            if(balls[bi].b_dict==1&&balls[bi].bx>x_e[balls[bi].plane_num]+5)balls[bi].b_dict=-1;
        }
```
balls[bi].bx += balls[bi].b_dict;
break;
}

} case 2: {
if(balls[bi].b_dict<=1& &balls[bi].by<y_s[balls[bi].plane_num]-5)balls[bi].b_dict=1;
if(balls[bi].b_dict==1& &balls[bi].by>y_e[balls[bi].plane_num]+5)balls[bi].b_dict=-1;
balls[bi].by += balls[bi].b_dict;
break;
}

case 3: {
if(balls[bi].b_dict==-1& &balls[bi].bz<z_s[balls[bi].plane_num]-5)balls[bi].b_dict=1;
if(balls[bi].b_dict==1& &balls[bi].bz>z_e[balls[bi].plane_num]+5)balls[bi].b_dict=-1;
balls[bi].bz += balls[bi].b_dict;
break;
}
}

for(kk=0;kk<cubnum;kk++){}
for(kk2=kk+1;kk2<cubnum;kk2++)
if(balls[kk].return_protect==0&&balls[kk2].return_protect==0)combine_ball(kk,kk2);
}

// write message
for(wi = 0;wi<4;wi++){
    message[40+wi*3] = balls[wi].bx;
    message[41+wi*3] = balls[wi].by-10;
    message[42+wi*3] = balls[wi].bz;
}
write_segments(message);

void ball_changeplane(int ballnum){
    if(balls[ballnum].next_is_edge == 0){
        balls[ballnum].next_jump_p = 0;
        balls[ballnum].prior_plane = balls[ballnum].plane_num;
    }
balls[ballnum].bx = (int)cp_node[balls[ballnum].jump_Node][0].x_node;
balls[ballnum].by = (int)cp_node[balls[ballnum].jump_Node][0].y_node;
balls[ballnum].bz = (int)cp_node[balls[ballnum].jump_Node][0].z_node;

if(extend_d[cp_node[balls[ballnum].jump_Node][0].plane_num] ==
extend_d[balls[ballnum].plane_num]){
    switch(extend_d[balls[ballnum].plane_num]){
        case 1:{if(((balls[ballnum].bx +
            balls[ballnum].prior_direct*5)!=(x_s[cp_node[balls[ballnum].jump_Node][0].plane_num]))&&((b
            alls[ballnum].bx +
            balls[ballnum].prior_direct*5)!=(x_e[cp_node[balls[ballnum].jump_Node][0].plane_num])))
            balls[ballnum].prior_direct = -balls[ballnum].b_dict;printf("%d",balls[ballnum].prior_direct);break;}
        case 2:{if(((balls[ballnum].by +
            balls[ballnum].prior_direct*5)!=(y_s[cp_node[balls[ballnum].jump_Node][0].plane_num]))&&((b
            alls[ballnum].by +
            balls[ballnum].prior_direct*5)!=(y_e[cp_node[balls[ballnum].jump_Node][0].plane_num])))
            balls[ballnum].prior_direct = -balls[ballnum].b_dict;printf("%d",balls[ballnum].prior_direct);break;}
        case 3:{if(((balls[ballnum].bz +
            balls[ballnum].prior_direct*5)!=(z_s[cp_node[balls[ballnum].jump_Node][0].plane_num]))&&((b
            alls[ballnum].bz +
            balls[ballnum].prior_direct*5)!=(z_e[cp_node[balls[ballnum].jump_Node][0].plane_num])))
            balls[ballnum].prior_direct = -balls[ballnum].b_dict;printf("%d",balls[ballnum].prior_direct);break;}
    } else
    balls[ballnum].prior_direct = balls[ballnum].b_dict;

    balls[ballnum].plane_num = cp_node[balls[ballnum].jump_Node][0].plane_num;
    if (balls[ballnum].jump_Node%2==0) {
        balls[ballnum].b_dict = 1;
    } else
    balls[ballnum].b_dict = -1;
    balls[ballnum].return_protect = 0;
} else if(balls[ballnum].next_is_edge == 1){
balls[ballnum].next_jump_p = 0;

    if(extend_d[balls[ballnum].edge_plane_num] == extend_d[balls[ballnum].plane_num])
    balls[ballnum].b_dict = -balls[ballnum].prior_direct;
else{
    balls[ballnum].b_dict = balls[ballnum].prior_direct;
}
balls[ballnum].prior_plane = balls[ballnum].plane_num;
balls[ballnum].plane_num = balls[ballnum].edge_plane_num;
balls[ballnum].bx = balls[ballnum].jump_x;
balls[ballnum].by = balls[ballnum].jump_y;
balls[ballnum].bz = balls[ballnum].jump_z;
balls[ballnum].next_is_edge = 0;
balls[ballnum].return_protect = 0;
}

// get the nearest jump point for ball
void calculate_jump_point(int ball_num){
    int i;
    int x_value, y_value, z_value;
    int first = 1;
    int diff;
    int diff_value1;
    int diff_value2;
    int pass = 0;
    diff = extend_d[balls[ball_num].plane_num];
balls[ball_num].next_jump_p = 0;
balls[ball_num].next_is_edge = 0;
    if (Plane_info[balls[ball_num].plane_num].connectpoint_num != 0){
        for (i = 0; i<Plane_info[balls[ball_num].plane_num].connectpoint_num; i++){
            //if the jumping point is in front of balls direct
            if( ((Plane_info[balls[ball_num].plane_num].P_x[i]-balls[ball_num].bx)/balls[ball_num].b_dict)>0)
            if( (Plane_info[balls[ball_num].plane_num].P_y[i]-balls[ball_num].by)/balls[ball_num].b_dict)>0)
            if( (Plane_info[balls[ball_num].plane_num].P_z[i]-balls[ball_num].bz)/balls[ball_num].b_dict)>0 ) {

                if(balls[ball_num].bx !=
                    Plane_info[balls[ball_num].plane_num].P_x[i]||balls[ball_num].by !=
                    Plane_info[balls[ball_num].plane_num].P_y[i]||balls[ball_num].bz !=
                    Plane_info[balls[ball_num].plane_num].P_z[i]){

                    //
                }
            }
        }
    }
balls[ball_num].next_jump_p = 1;
}
if (first==1) {
    first = 0;
    balls[ball_num].jump_Node = Plane_info[balls[ball_num].plane_num].N_connect[i];
    balls[ball_num].jump_x = Plane_info[balls[ball_num].plane_num].P_x[i];
    balls[ball_num].jump_y = Plane_info[balls[ball_num].plane_num].P_y[i];
    balls[ball_num].jump_z = Plane_info[balls[ball_num].plane_num].P_z[i];
    switch (diff) {
        case 1:
            diff_value1 = Plane_info[balls[ball_num].plane_num].P_x[i] - balls[ball_num].bx;
            break;
        case 2:
            diff_value1 = Plane_info[balls[ball_num].plane_num].P_y[i] - balls[ball_num].by;
            break;
        case 3:
            diff_value1 = Plane_info[balls[ball_num].plane_num].P_z[i] - balls[ball_num].bz;
            break;
    }
    diff_value2 = diff_value1;
}
else{
    switch (diff) {
        case 1:
            diff_value1 = Plane_info[balls[ball_num].plane_num].P_x[i] - balls[ball_num].bx;
            break;
        case 2:
            diff_value1 = Plane_info[balls[ball_num].plane_num].P_y[i] - balls[ball_num].by;
            break;
        case 3:
            diff_value1 = Plane_info[balls[ball_num].plane_num].P_z[i] - balls[ball_num].bz;
            break;
    }
    default:
        break;
}
    diff_value2 = diff_value1;
}
balls[ball_num].bz;
    break;
  }
default:
  break;
}
if (diff_value1<diff_value2) {
  balls[ball_num].jump_Node = Plane_info[balls[ball_num].plane_num].N_connect[i];
  balls[ball_num].jump_x = Plane_info[balls[ball_num].plane_num].P_x[i];
  balls[ball_num].jump_y = Plane_info[balls[ball_num].plane_num].P_y[i];
  balls[ball_num].jump_z = Plane_info[balls[ball_num].plane_num].P_z[i];
}
}
}
if(balls[ball_num].next_jump_p == 0){
  if(Plane_info[balls[ball_num].plane_num].edge_node_connect[0]==1&&balls[ball_num].b_dict==-1){
    balls[ball_num].next_is_edge = 1;
    balls[ball_num].jump_x = Plane_info[balls[ball_num].plane_num].edge_node_jumpx[0];
    balls[ball_num].jump_y = Plane_info[balls[ball_num].plane_num].edge_node_jumpy[0];
    balls[ball_num].jump_z = Plane_info[balls[ball_num].plane_num].edge_node_jumpz[0];
    balls[ball_num].edge_plane_num = Plane_info[balls[ball_num].plane_num].edge_node_jump_plane[0];
    balls[ball_num].next_jump_p = 1;
  }
  else if(Plane_info[balls[ball_num].plane_num].edge_node_connect[1]==1&&balls[ball_num].b_dict==1){
    balls[ball_num].next_is_edge = 1;
    balls[ball_num].jump_x = Plane_info[balls[ball_num].plane_num].edge_node_jumpx[1];
    balls[ball_num].jump_y = Plane_info[balls[ball_num].plane_num].edge_node_jumpy[1];
    balls[ball_num].jump_z = Plane_info[balls[ball_num].plane_num].edge_node_jumpz[1];

```c
balls[ball_num].next_jump_p = 1;
}
}
}

/* Read and print the segment values */
void print_segment_info() {
    vga_led_arg_t vla;
    int i;

    for (i = 0; i < VGA_LED_DIGITS; i++) {
        vla.digit = i;
        if (ioctl(vga_led_fd, VGA_LED_READ_DIGIT, &vla)) {
            perror("ioctl(VGA_LED_READ_DIGIT) failed");
            return;
        }
    }
}

/* Write the contents of the array to the display */
void write_segments(const unsigned char segs[64]) {
    vga_led_arg_t vla;
    int i;
    for (i = 0; i < VGA_LED_DIGITS; i++) {
        vla.digit = i;
        vla.segments = segs[i];
        if (ioctl(vga_led_fd, VGA_LED_WRITE_DIGIT, &vla)) {
            perror("ioctl(VGA_LED_WRITE_DIGIT) failed");
            return;
        }
    }
}

int main() {

    /* Variable definition */
    int i;
    static const char filename[] = "/dev/vga_led";
    int operation;
```
int err, col;
struct usb_keyboard_packet packet;
int transferred;
char keystate[12];

x_angle_l = 0;
y_angle_l = 0;
x_angle_h = 0;
y_angle_h = 0;

memcpy(&extend_d, &extend_d1, 20);
memcpy(&x_s, &x_s1, 20);
memcpy(&x_e, &x_e1, 20);
memcpy(&y_s, &y_s1, 20);
memcpy(&y_e, &y_e1, 20);
memcpy(&z_s, &z_s1, 20);
memcpy(&z_e, &z_e1, 20);

vga_led_arg_t vla;

/*  Open the keyboard  */
if ( (keyboard = openkeyboard(&endpoint_address)) == NULL ) {
    fprintf(stderr, "Did not find a keyboard\n");
    exit(1);
}

//printf("VGA LED Userspace program started\n");
if ( (vga_led_fd = open(filename, O_RDWR)) == -1 ) {
    fprintf(stderr, "could not open %s\n", filename);
    return -1;
}

//printf("initial state: ");
print_segment_info();
write_segments(message);
//printf("current state: ");
print_segment_info();
//printf("VGA LED Userspace program started\n");
if ( (vga_led_fd = open(filename, O_RDWR)) == -1 ) {
    fprintf(stderr, "could not open %s\n", filename);
    return -1;
}

//printf("initial state: ");
print_segment_info();
write_segments(message);
//printf("current state: ");
print_segment_info();

change_level:
printf("initialize\n");
initialize();
operation = 3;rotation_rate = 2;for(i=0;i<25600000;i++){if(i%100000 == 0)transfervalue (operation);}flag = 1;
operation = 1;rotation_rate = 2;for(i=0;i<25600000;i++){if(i%100000 == 0)transfervalue (operation);}flag = 1;
transfervalue(5);

/* Look for and handle keypresses */
for (;;) {
message[6] = 0x01;
bball_moving();
ooperation = 0;
lusbus_interrupt_transfer(keyboard, endpoint_address,
    (unsigned char *) &packet, sizeof(packet),
    &transferred, 0);
    if (transferred == sizeof(packet)) {
        if(packet.keycode[3] == 0x06&&packet.keycode[4] == 0x00){operation = 1;rotation_rate = 1;}
        else if(packet.keycode[3] == 0x02&&packet.keycode[4] == 0x00){operation = 2;rotation_rate = 1;}
        else if(packet.keycode[3] == 0x00&&packet.keycode[4] == 0x00){operation = 3;rotation_rate = 1;}
        else if(packet.keycode[3] == 0x04&&packet.keycode[4] == 0x00){operation = 4;rotation_rate = 1;}
        else if(packet.keycode[3] == 0x16&&packet.keycode[4] == 0x00){operation = 1;rotation_rate = 4;}
        else if(packet.keycode[3] == 0x12&&packet.keycode[4] == 0x00){operation = 2;rotation_rate = 4;}
        else if(packet.keycode[3] == 0x10&&packet.keycode[4] == 0x00){operation = 3;rotation_rate = 4;}
        else if(packet.keycode[3] == 0x14&&packet.keycode[4] == 0x00){operation = 4;rotation_rate = 4;}
        else if(packet.keycode[3] == 0x4f&&packet.keycode[4] == 0x00){operation = 5;}
        else if(packet.keycode[4] == 0x04&&packet.keycode[3] == 0x0f&&flag == 0)
        {operation = 1;rotation_rate = 2;for(i=0;i<5120000;i++){if(i%100000 == 0)transfervalue (operation);}flag = 1;}
        else if(packet.keycode[4] == 0x08&&packet.keycode[3] == 0x0f&&flag == 0)
{operation = 2; rotation_rate = 2; for(i=0;i<5120000;i++) if(i%10000 == 0) transfervalue(operation); } flag = 1;

else if(packet.keycode[3] == 0x1f&&packet.keycode[4] == 0x0c){operation = 6;}
else if(packet.keycode[3] == 0x2f&&packet.keycode[4] == 0x0c){operation = 7;}
else if(packet.keycode[3] == 0x4f&&packet.keycode[4] == 0x0c){operation = 8;}
else if(packet.keycode[3] == 0x8f&&packet.keycode[4] == 0x0c){operation = 9;}
else if(packet.keycode[3] == 0x0f&&packet.keycode[4] == 0x0d){operation = 10;}
else if(packet.keycode[3] == 0x0f&&packet.keycode[4] == 0x0e){operation = 11;}

if(packet.keycode[3] == 0x0f&&packet.keycode[4] == 0x00) flag=0;

if(operation == 6 || operation == 7 || operation == 8 || operation == 9 || operation == 10 || operation == 11){
    switch(operation)
    {
        case 6:
            
            memcpy(&extend_d, &extend_d1, 20);
            memcpy(&x_s, &x_s1, 20);
            memcpy(&x_e, &x_e1, 20);
            memcpy(&y_s, &y_s1, 20);
            memcpy(&y_e, &y_e1, 20);
            memcpy(&z_s, &z_s1, 20);
            memcpy(&z_e, &z_e1, 20);
            break;
        
        case 7:
            
            memcpy(&extend_d, &extend_d2, 20);
            memcpy(&x_s, &x_s2, 20);
            memcpy(&x_e, &x_e2, 20);
            memcpy(&y_s, &y_s2, 20);
            memcpy(&y_e, &y_e2, 20);
            memcpy(&z_s, &z_s2, 20);
            memcpy(&z_e, &z_e2, 20);
            break;
        
        case 8:
            
            memcpy(&extend_d, &extend_d3, 20);
            memcpy(&x_s, &x_s3, 20);
            memcpy(&x_e, &x_e3, 20);
memcpy(&y_s, &y_s3, 20);
memcpy(&y_e, &y_e3, 20);
memcpy(&z_s, &z_s3, 20);
memcpy(&z_e, &z_e3, 20);
break;
}
case 9:
{
memcpy(&extend_d, &extend_d4, 20);
memcpy(&x_s, &x_s4, 20);
memcpy(&x_e, &x_e4, 20);
memcpy(&y_s, &y_s4, 20);
memcpy(&y_e, &y_e4, 20);
memcpy(&z_s, &z_s4, 20);
memcpy(&z_e, &z_e4, 20);
break;
}
case 10:
{
memcpy(&extend_d, &extend_d5, 20);
memcpy(&x_s, &x_s5, 20);
memcpy(&x_e, &x_e5, 20);
memcpy(&y_s, &y_s5, 20);
memcpy(&y_e, &y_e5, 20);
memcpy(&z_s, &z_s5, 20);
memcpy(&z_e, &z_e5, 20);
break;
}
case 11:
{
memcpy(&extend_d, &extend_d6, 20);
memcpy(&x_s, &x_s6, 20);
memcpy(&x_e, &x_e6, 20);
memcpy(&y_s, &y_s6, 20);
memcpy(&y_e, &y_e6, 20);
memcpy(&z_s, &z_s6, 20);
memcpy(&z_e, &z_e6, 20);
initialize();
operation = 3;rotation_rate = 2;for(i=0;i<25600000;i++){if(i%100000 == 0)transfervalue(operation);}flag = 1;
operation = 1;rotation_rate = 2;for(i=0;i<25600000;i++){if(i%100000 == 0)transfervalue(operation);}flag = 1;
custom_design(packet,transferred);
break;
}
break;
}
goto change_level;
}

if(operation == 1 || operation == 2 || operation == 3 || operation == 4 || operation == 5){
transfervalue(operation);
calculate_pass();
message[5] = (2*(2*(2*(0 + cubic[4]) + cubic[3]) + cubic[2]) + cubic[1]) + cubic[0]);
write_segments(message);
}
return 0;
}

void custom_design(struct usb_keyboard_packet packet, int transferred){
    int x, y, z;
    int flag = 0;
    int length;
    int orientation;
    int i;
    for(i = 0; i<5; i++){
        x=0;
        y=0;
        z=0;
        length = 5;
        orientation = 0;
        wait:
            libusb_interrupt_transfer(keyboard, endpoint_address,
            (unsigned char *) &packet, sizeof(packet),
            &transferred, 0);
        if(!(packet.keycode[3] == 0x0f&&packet.keycode[4] == 0x00)) goto wait;
        while(packet.keycode[3] != 0x0f||packet.keycode[4] != 0x02){
            generate_cubic();

            if(packet.keycode[3] == 0x0f&&packet.keycode[4] == 0x00)flag=0;
            else if(packet.keycode[4] == 0x04&&packet.keycode[3] == 0x0f&flag ==
0)(orientation = 0;)
            else if(packet.keycode[4] == 0x08&&packet.keycode[3] == 0x0f&flag ==
0){orientation = 1;)

    else if(packet.keycode[3] == 0x1f&&packet.keycode[4] == 0x00&&flag == 0){
        if(length != 125) length = length + 5;
        flag = 1;
    }

    else if(packet.keycode[3] == 0x4f&&packet.keycode[4] == 0x00&&flag == 0){
        if(length != 5) length = length - 5;
        flag = 1;
    }

    else if(packet.keycode[3] == 0x00&&packet.keycode[4] == 0x00&&flag == 0){
        if(y != -100) y = y - 10;
        flag = 1;
    }

    else if(packet.keycode[3] == 0x04&&packet.keycode[4] == 0x00&&flag == 0){
        if(y != 100) y = y + 10;
        flag = 1;
    }

    else if(packet.keycode[3] == 0x06&&packet.keycode[4] == 0x00&&flag == 0){
        if(orientation == 0&&x != -100) x = x - 10;
        else if(orientation == 1&&z != -100) z = z - 10;
        flag = 1;
    }

    else if(packet.keycode[3] == 0x02&&packet.keycode[4] == 0x00&&flag == 0){
        if(orientation == 0&&x != 100) x = x + 10;
        else if(orientation == 1&&z != 100) z = z + 10;
        flag = 1;
    }
}

if(orientation == 0){
    extend_d[i] = 3;
    x_s[i] = x - width;
    x_e[i] = x + width;
    y_s[i] = y - width;
    y_e[i] = y + width;
    z_s[i] = z - length;
    z_e[i] = z + length;
} else{
    extend_d[i] = 1;
    x_s[i] = x - length;
    x_e[i] = x + length;
    y_s[i] = y - width;
\[ y_{e[i]} = y + \text{width}; \]
\[ z_{s[i]} = z - \text{width}; \]
\[ z_{e[i]} = z + \text{width}; \]

```c
initialize_endPoint();
initialize_ball();

libusb_interrupt_transfer(keyboard, endpoint_address,
    (unsigned char *) &packet, sizeof(packet),
    &transferred, 0);
```

```c
generate_cubic()
{
    int i,j,k;
    k = 10;
    for(i = 0; i<cu;
```
message[0] = 0x00;
message[1] = 0x00;
message[2] = 0x00;
message[3] = 0x00;
message[5] = 0x00;
message[6] = 0x00;
x_angle_l = 0;
y_angle_l = 0;
x_angle_h = 0;
y_angle_h = 0;
generate_cubic();

int i,j,k;

for(i = 0;i<cubnum;i++){
    Plane_info[i].edge_node[0] = 2*i;
    Plane_info[i].edge_node[1] = 2*i + 1;
}

initialize_ball();
initialize_endPoint();
for (i = 0; i<cubnum; i++) {
    switch (extend_d[i]) {
    case 1:
        cubic_nodes_num[i] = ((x_e[i] - x_s[i])/10 - 2)*10 + 10;
        break;
    case 2:
        cubic_nodes_num[i] = ((y_e[i] - y_s[i])/10 - 2)*10 + 10;
        break;
    case 3:
        cubic_nodes_num[i] = ((z_e[i] - z_s[i])/10 - 2)*10 + 10;
        break;
    default:
        break;
    }
}

//generate all comparing nodes, stored in structure cp_node[cubnum*2][cubnum]
for (i = 0; i<2*cubnum; i++) {
    for (j = 0; j<cubnum; j++) {
        cp_node[i][j].plane_num = i/2;
switch (extend_d[i/2]) {
    case 1:
        if (i % 2 == 0)
            cp_node[i][j].x_node = x_s[i/2] - width;
        else
            cp_node[i][j].x_node = x_e[i/2] + width;
        cp_node[i][j].y_node = y_s[i/2] + width;
        cp_node[i][j].z_node = z_s[i/2] + width;
        break;
    case 2:
        if (i % 2 == 0)
            cp_node[i][j].y_node = y_s[i/2] - width;
        else
            cp_node[i][j].y_node = y_e[i/2] + width;
        cp_node[i][j].x_node = x_s[i/2] + width;
        cp_node[i][j].z_node = z_s[i/2] + width;
        break;
    case 3:
        if (i % 2 == 0)
            cp_node[i][j].z_node = z_s[i/2] - width;
        else
            cp_node[i][j].z_node = z_e[i/2] + width;
        cp_node[i][j].y_node = y_s[i/2] + width;
        cp_node[i][j].x_node = x_s[i/2] + width;
        break;
    default:
        break;
}
if (cp_node[i][j].plane_num != j)
for (k = 0; k<=cubic_nodes_num[j]; k++) {
    switch (extend_d[j]) {
    case 1:
        cp_node[i][j].x_va[k] = x_s[j] + 5 + k - cp_node[i][j].x_node;
        cp_node[i][j].y_va[k] = y_s[j] + 5 - cp_node[i][j].y_node;
        cp_node[i][j].z_va[k] = z_s[j] + 5 - cp_node[i][j].z_node;
        D = cp_node[i][j].x_va[k]*cp_node[i][j].x_va[k] +
            cp_node[i][j].y_va[k]*cp_node[i][j].y_va[k] +
            cp_node[i][j].z_va[k]*cp_node[i][j].z_va[k];
D = sqrt(D);
cp_node[i][j].x_va[k] = cp_node[i][j].x_va[k]/D;
cp_node[i][j].y_va[k] = cp_node[i][j].y_va[k]/D;
cp_node[i][j].z_va[k] = cp_node[i][j].z_va[k]/D;
break;
}
case 2:{
cp_node[i][j].x_va[k] = x_s[j] + 5 - cp_node[i][j].x_node;
cp_node[i][j].y_va[k] = y_s[j] + 5 + k - cp_node[i][j].y_node;
cp_node[i][j].z_va[k] = z_s[j] + 5 - cp_node[i][j].z_node;
D = cp_node[i][j].x_va[k]*cp_node[i][j].x_va[k] + cp_node[i][j].y_va[k]*cp_node[i][j].y_va[k] + cp_node[i][j].z_va[k]*cp_node[i][j].z_va[k];
D = sqrt(D);
cp_node[i][j].x_va[k] = cp_node[i][j].x_va[k]/D;
cp_node[i][j].y_va[k] = cp_node[i][j].y_va[k]/D;
cp_node[i][j].z_va[k] = cp_node[i][j].z_va[k]/D;
break;
}
case 3:{
cp_node[i][j].x_va[k] = x_s[j] + 5 - cp_node[i][j].x_node;
cp_node[i][j].y_va[k] = y_s[j] + 5 - cp_node[i][j].y_node;
cp_node[i][j].z_va[k] = z_s[j] + 5 + k - cp_node[i][j].z_node;
D = cp_node[i][j].x_va[k]*cp_node[i][j].x_va[k] + cp_node[i][j].y_va[k]*cp_node[i][j].y_va[k] + cp_node[i][j].z_va[k]*cp_node[i][j].z_va[k];
D = sqrt(D);
cp_node[i][j].x_va[k] = cp_node[i][j].x_va[k]/D;
cp_node[i][j].y_va[k] = cp_node[i][j].y_va[k]/D;
cp_node[i][j].z_va[k] = cp_node[i][j].z_va[k]/D;
break;
}
default:
    break;
}
int ci, cj, ck;
int bi;
rotate_angle_x = 2*3.14159*(x_angle_l + x_angle_h*256)/4096;
rotate_angle_y = 2*3.14159*(y_angle_l + y_angle_h*256)/4096;

initialize_plane();
v_x = 0.0;
v_y = 0.0;
v_z = 1.0;

v_x1 = v_x;
v_y1 = cos(rotate_angle_y)*v_y + sin(rotate_angle_y)*v_z;
v_z1 = -sin(rotate_angle_y)*v_y + cos(rotate_angle_y)*v_z;

v_x2 = -sin(rotate_angle_x)*v_z1 + cos(rotate_angle_x)*v_x1;
v_y2 = v_y1;
v_z2 = cos(rotate_angle_x)*v_z1 + sin(rotate_angle_x)*v_x1;

for (cj = 0; cj<cubnum; cj++) {
    cubic[cj] = 0;
}
for (ci=0;ci<2*cubnum;ci++)
    for (cj = 0; cj<cubnum; cj++) {
        if ((cubic[cj]! =1) || (cubic[cp_node[ci][cj].plane_num] != 1))
            for (ck = 0; ck<=cubic_nodes_num[cj]; ck++) {
                if (cubic[cj] ==1 && cubic[cp_node[ci][cj].plane_num] == 1) break;
                if (cp_node[ci][cj].plane_num !=(cj)) {
                    D2 = sqrt((v_x2-cp_node[ci][cj].x_va[ck])*(v_x2-cp_node[ci][cj].x_va[ck]) + (v_y2-cp_node[ci][cj].y_va[ck])*(v_y2-cp_node[ci][cj].y_va[ck]) + (v_z2-cp_node[ci][cj].z_va[ck])*(v_z2-cp_node[ci][cj].z_va[ck])) < sqrt((v_x2+cp_node[ci][cj].x_va[ck])*(v_x2+cp_node[ci][cj].x_va[ck]) + (v_y2+cp_node[ci][cj].y_va[ck])*(v_y2+cp_node[ci][cj].y_va[ck]) + (v_z2+cp_node[ci][cj].z_va[ck])*(v_z2+cp_node[ci][cj].z_va[ck])?
                        sqrt((v_x2-cp_node[ci][cj].x_va[ck])*(v_x2-cp_node[ci][cj].x_va[ck]) + (v_y2-cp_node[ci][cj].y_va[ck])*(v_y2-cp_node[ci][cj].y_va[ck]) + (v_z2-cp_node[ci][cj].z_va[ck])*(v_z2-cp_node[ci][cj].z_va[ck]));
                        switch(extend_d[ci/2]){
                            case 1:

```
D3 = (v_x2-1)*(v_x2-1) + (v_y2-0)*(v_y2-0) + (v_z2-0)*(v_z2-0);
break;
}
case 2:
    D3 = (v_x2-0)*(v_x2-0) + (v_y2-1)*(v_y2-1) + (v_z2-0)*(v_z2-0);
    break;
}
case 3:
    D3 = (v_x2-0)*(v_x2-0) + (v_y2-0)*(v_y2-0) + (v_z2-1)*(v_z2-1);
    break;
}

seeSurface1 = (ci%2 ? D3 > 2 : D3 < 2) && (sqrt((v_x2-cp_node[ci][cj].x_va[ck])*(v_x2-cp_node[ci][cj].x_va[ck]) + (v_y2-cp_node[ci][cj].y_va[ck])*(v_y2-cp_node[ci][cj].y_va[ck]) +
    (v_z2-cp_node[ci][cj].z_va[ck])*(v_z2-cp_node[ci][cj].z_va[ck]))<sqrt((v_x2+cp_node[ci][cj].x_va[ck])*(v_x2+cp_node[ci][cj].x_va[ck]) +
    (v_y2+cp_node[ci][cj].y_va[ck])*(v_y2+cp_node[ci][cj].y_va[ck]) +
    (v_z2+cp_node[ci][cj].z_va[ck])*(v_z2+cp_node[ci][cj].z_va[ck])));
nonSeeSurface2 = !(ci%2 ? D3 > 2 : D3 < 2) &&
    (sqrt((v_x2-cp_node[ci][cj].x_va[ck])*(v_x2-cp_node[ci][cj].x_va[ck]) + (v_y2-cp_node[ci][cj].y_va[ck])*(v_y2-cp_node[ci][cj].y_va[ck]) +
    (v_z2-cp_node[ci][cj].z_va[ck])*(v_z2-cp_node[ci][cj].z_va[ck]))>sqrt((v_x2+cp_node[ci][cj].x_va[ck])*(v_x2+cp_node[ci][cj].x_va[ck]) +
    (v_y2+cp_node[ci][cj].y_va[ck])*(v_y2+cp_node[ci][cj].y_va[ck]) +
    (v_z2+cp_node[ci][cj].z_va[ck])*(v_z2+cp_node[ci][cj].z_va[ck])));
parallel = (extend_d[cj/2] == extend_d[cj]) && (sqrt((v_x2-cp_node[ci][cj].x_va[ck])*(v_x2-cp_node[ci][cj].x_va[ck]) + (v_y2-cp_node[ci][cj].y_va[ck])*(v_y2-cp_node[ci][cj].y_va[ck]) +
    (v_z2-cp_node[ci][cj].z_va[ck])*(v_z2-cp_node[ci][cj].z_va[ck]))>sqrt((v_x2+cp_node[ci][cj].x_va[ck])*(v_x2+cp_node[ci][cj].x_va[ck]) +
    (v_y2+cp_node[ci][cj].y_va[ck])*(v_y2+cp_node[ci][cj].y_va[ck]) +
    (v_z2+cp_node[ci][cj].z_va[ck])*(v_z2+cp_node[ci][cj].z_va[ck])));
if (D2<precision && !seeSurface1 && !nonSeeSurface2 && !parallel) {
    cubic[cp_node[ci][cj].plane_num] = 1;
    cubic[cj] = 1;
    record_location1x = (int)cp_node[ci][cj].x_node;
    record_location1y = (int)cp_node[ci][cj].y_node;
    record_location1z = (int)cp_node[ci][cj].z_node;
    switch(extend_d[cj]){
        case 1:{
            record_location2x = (int)x_s[cj] + 5 +ck;
            record_location2y = (int)y_s[cj] + 5;
record_location2z = (int)z_s[cj] + 5;
break;
}
case 2:
    record_location2x = (int)x_s[cj] + 5;
    record_location2y = (int)y_s[cj] + 5 +ck;
    record_location2z = (int)z_s[cj] + 5;
    break;
}
case 3:
    record_location2x = (int)x_s[cj] + 5;
    record_location2y = (int)y_s[cj] + 5;
    record_location2z = (int)z_s[cj] + 5 +ck;
    break;
}
}

Plane_info[ci/2].edge_node_connect[ci%2] = 1;
Plane_info[ci/2].edge_node_jumpx[ci%2] = record_location2x;
Plane_info[ci/2].edge_node_jumpy[ci%2] = record_location2y;
Plane_info[ci/2].edge_node_jumpz[ci%2] = record_location2z;
Plane_info[ci/2].edge_node_jump_plane[ci%2] = cj;

Plane_info[cj].N_connect[Plane_info[cj].connectpoint_num] = ci;
Plane_info[cj].P_x[Plane_info[cj].connectpoint_num] = record_location2x;
Plane_info[cj].P_y[Plane_info[cj].connectpoint_num] = record_location2y;
Plane_info[cj].P_z[Plane_info[cj].connectpoint_num] = record_location2z;
Plane_info[cj].connectpoint_num++;
}
}

for(bi = 0; bi<cubnum-1;bi++)calculate_jump_point(bi);
void transfervalue(int op)
{
    int p;
    for(p =0;p<cubnum - 1; p++){
        if(balls[p].return_protect == 1){
            balls[p].bx = balls[p].ret_bx;
            balls[p].by = balls[p].ret_by;
            balls[p].bz = balls[p].ret_bz;
            balls[p].plane_num = balls[p].ret_direct;
            balls[p].return_protect = 0;
        }
    }
    switch(op)
    {
    case 1:
    {
        if(x_angle_l >= 256 - rotation_rate){
            x_angle_l = x_angle_l + rotation_rate - 256;
            message[0] = message[0] + rotation_rate;
            if(x_angle_h == 15)
            {
                x_angle_h = 0;
                message[1] = 0x00;
            }
            else
            {
                x_angle_h++;
            }
        }
        else
        {
            x_angle_l+=rotation_rate;
            message[0] = message[0]+rotation_rate;
        }
        break;
    }
    case 2:
    {
        if(x_angle_l <= rotation_rate - 1)
        {
            x_angle_l = x_angle_l + 256 - rotation_rate;
            message[0] = message[0] - rotation_rate;
        }
    }
    }
}
if(x_angle_h == 0)
{
    x_angle_h = 15;
    message[1] = 0x0F;
}
else
{
    x_angle_h--;
    message[1] = message[1]-1;
}

else
{
    x_angle_l-=rotation_rate;
    message[0] = message[0]-rotation_rate;
}
break;

case 3:
{

    if(y_angle_l>=256 - rotation_rate)
    {
        y_angle_l = y_angle_l + rotation_rate - 256;
        if(y_angle_h == 15)
        {
            y_angle_h = 0;
            message[3] = 0x00;
        }
        else
        {
            y_angle_h++;
        }
    }
    else
    {
        y_angle_l+=rotation_rate;
    }
    break;
}

case 4:
{ if(y_angle_l<=rotation_rate - 1){
    y_angle_l = y_angle_l + 256 - rotation_rate;
    if(y_angle_h == 0)
    {
        y_angle_h = 15;
        message[3] = 0x0F;
    }
    else
    {
        y_angle_h--;
    }
} else
{
    y_angle_l-=rotation_rate;
} break;
}

case 5: {
    ball_reach[0] = 0;
    ball_reach[1] = 0;
    ball_reach[2] = 0;
    ball_reach[3] = 0;
    initialize_ball();
    break;
} break;
}
write_segments(message);
if(message[4] = 0x01)message[4] = 0x00;
return NULL;
}

void combine_ball(int i, int j){
if(abs(balls[i].bx - balls[j].bx) <= 1 && abs(balls[i].by - balls[j].by) <= 1 && abs(balls[i].bz - balls[j].bz) <= 1)
{
    balls[j].b_dict = balls[i].b_dict;
    balls[j].next_jump_p = balls[i].next_jump_p;
    balls[j].jump_x = balls[i].jump_x;
    balls[j].jump_y = balls[i].jump_y;
    balls[j].jump_z = balls[i].jump_z;
    balls[j].jump_Node = balls[i].jump_Node;
    balls[j].prior_plane = balls[i].prior_plane;
    balls[j].prior_direct = balls[i].prior_direct;
    balls[j].next_is_edge = balls[i].next_is_edge;
    balls[j].edge_plane_num = balls[i].edge_plane_num;
    balls[j].ret_bx = balls[i].ret_bx;
    balls[j].ret_by = balls[i].ret_by;
    balls[j].ret_bz = balls[i].ret_bz;
    balls[j].return_protect = balls[i].return_protect;
    balls[j].ret_direct = balls[i].ret_direct;
}

Vga_led.c

/*
 * Device driver for the VGA LED Emulator
 * 
 * A Platform device implemented using the misc subsystem
 * 
 * Stephen A. Edwards
 * Columbia University
 * 
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *       drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 * 
 * "make" to build
 * insmod vga_led.ko
 * 
 * Check code style with
 * checkpatch.pl --file --no-tree vga_led.c
 */
#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "vga_led.h"

#define DRIVER_NAME "vga_led"

/*@ Information about our devic dsfhsdkfaskjfgdsakj */
struct vga_led_dev {
    struct resource res; /* Resource: our registers */
    void __iomem *virtbase; /* Where registers can be accessed in memory */
    int segments[VGA_LED_DIGITS];
} dev;

/*@ Write segments of a single digit */
/* Assumes digit is in range and the device information has been set up */
static void write_digit(int digit, int segments)
{
    iowrite8(segments, dev.virtbase + digit);
    dev.segments[digit] = segments;
}

/*@ Handle ioctl() calls from userspace: */
/* Read or write the segments on single digits. */
/* Note extensive error checking of arguments */
static long vga_led_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{

vga_led_arg_t vla;

switch (cmd) {
    case VGA_LED_WRITE_DIGIT:
        if (copy_from_user(&vla, (vga_led_arg_t *) arg,
                           sizeof(vga_led_arg_t)))
            return -EACCES;
        if (vla.digit > 64) // modified by ly
            return -EINVAL;
        write_digit(vla.digit, vla.segments);
        break;
    case VGA_LED_READ_DIGIT:
        if (copy_from_user(&vla, (vga_led_arg_t *) arg,
                           sizeof(vga_led_arg_t)))
            return -EACCES;
        if (vla.digit > 64) // modified by ly
            return -EINVAL;
        vla.segments = dev.segments[vla.digit];
        if (copy_to_user((vga_led_arg_t *) arg, &vla,
                          sizeof(vga_led_arg_t)))
            return -EACCES;
        break;
    default:
        return -EINVAL;
}

return 0;

/* The operations our device knows how to do */
static const struct file_operations vga_led_fops = {
    .owner = THIS_MODULE,
    .unlocked_ioctl = vga_led_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice vga_led_misc_device = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = DRIVER_NAME,
    .fops = &vga_led_fops,
};
/* Initialization code: get resources (registers) and display */
static int __init vga_led_probe(struct platform_device *pdev)
{
    static unsigned char welcome_message[VGA_LED_DIGITS] = {
        0x3E, 0x7D, 0x77, 0x08, 0x38, 0x79, 0x5E, 0x00,
        0x3E, 0x7D, 0x77, 0x08, 0x38, 0x79, 0x5E, 0x00};
    int i, ret;

    /* Register ourselves as a misc device: creates /dev/vga_led */
    ret = misc_register(&vga_led_misc_device);

    /* Get the address of our registers from the device tree */
    ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
    if (ret) {
        ret = -ENOENT;
        goto out_deregister;
    }

    /* Make sure we can use these registers */
    if (request_mem_region(dev.res.start, resource_size(&dev.res),
        DRIVER_NAME) == NULL) {
        ret = -EBUSY;
        goto out_deregister;
    }

    /* Arrange access to our registers */
    dev.virtbase = of_iomap(pdev->dev.of_node, 0);
    if (dev.virtbase == NULL) {
        ret = -ENOMEM;
        goto out_release_mem_region;
    }

    /* Display a welcome message */
    for (i = 0; i < VGA_LED_DIGITS; i++)
        write_digit(i, welcome_message[i]);

    out_deregister:
        misc_deregister(&vga_led_misc_device);
    out_release_mem_region:
    return ret;
}
return 0;

out_release_mem_region:
    release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
    misc_deregister(&vga_led_misc_device);
    return ret;
}

/* Clean-up code: release resources */
static int vga_led_remove(struct platform_device *pdev)
{
    iounmap(dev.virtbase);
    release_mem_region(dev.res.start, resource_size(&dev.res));
    misc_deregister(&vga_led_misc_device);
    return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id vga_led_of_match[] = {
    {.compatible = "altr,vga_led" },
    {},
};
MODULE_DEVICE_TABLE(of, vga_led_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver vga_led_driver = {
    .driver = {
        .name = DRIVER_NAME,
        .owner = THIS_MODULE,
        .of_match_table = of_match_ptr(vga_led_of_match),
    },
    .remove = __exit_p(vga_led_remove),
};

/* Called when the module is loaded: set things up */
static int __init vga_led_init(void)
{
    pr_info(DRIVER_NAME "\n");
    return platform_driver_probe(&vga_led_driver, vga_led_probe);
}
/* Called when the module is unloaded: release resources */
static void __exit vga_led_exit(void)
{
    platform_driver_unregister(&vga_led_driver);
    pr_info(DRIVER_NAME "exit\n");
}

module_init(vga_led_init);
module_exit(vga_led_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("VGA 7-segment LED Emulator");

Vga_led.h

#ifndef _VGA_LED_H
#define _VGA_LED_H

#include <linux/ioctl.h>

#define VGA_LED_DIGITS 64

typedef struct {
    unsigned char digit;  /* 0, 1, .. , VGA_LED_DIGITS - 1 */
    unsigned char segments; /* LSB is segment a, MSB is decimal point */
} vga_led_arg_t;

#define VGA_LED_MAGIC 'q'

/* ioctl's and their arguments */
#define VGA_LED_WRITE_DIGIT  _IOW(VGA_LED_MAGIC, 1, vga_led_arg_t *)
#define VGA_LED_READ_DIGIT  _IOWR(VGA_LED_MAGIC, 2, vga_led_arg_t *)

#endif

Makefile

CFLAGS = -Wall

OBJECTS = hello.o usbkeyboard.o

TARFILES = Makefile hello.c \

usbkeyboard.h usbkeyboard.c

hello : $(OBJECTS)
    cc $(CFLAGS) -o hello $(OBJECTS) -lusb -1.0 -pthread -lm

hello.tar.gz : $(TARFILES)
    rm -rf hello
    mkdir hello
    ln $(TARFILES) hello
    tar zcf hello.tar.gz hello
    rm -rf hello

hello.o : hello.c usbkeyboard.h
usbkeyboard.o : usbkeyboard.c usbkeyboard.h -lm

.PHONY : clean
clean :
    rm -rf *.o hello

Hardware

Miffilegen.m

function [outfname, rows, cols] = miffilegen_v2(infile,outfname, numrows, numcols)

    img = imread(infile);

    imgresized = imresize(img, [numrows numcols]);

    [rows, cols, rgb] = size(imgresized);

    imgscaled = imgresized;
    imshow(imgscaled);

    fid = fopen(outfname,'
');

    fprintf(fid,'-- %3ux%3u 24bit image color values\n\n',rows,cols);
    fprintf(fid,'WIDTH = 24;\n');
    fprintf(fid,'DEPTH = %4u;\n',rows*cols);
    fprintf(fid,'ADDRESS_RADIX = UNS;\n');
    fprintf(fid,'DATA_RADIX = UNS;\n');
    fprintf(fid,'CONTENT BEGIN\n');
count = 0;
for r = 1:rows
    for c = 1:cols
        red = uint32(imgscaled(r,c,1));
        green = uint32(imgscaled(r,c,2));
        blue = uint32(imgscaled(r,c,3));
        color = red*(65536) + green*(256) + blue;
        fprintf(fid,'%4u : %u;
        n',count, color);
        count = count + 1;
    end
end
fprintf(fid,'END;');
fclose(fid);

Sin_Cos_LUT.m

clear;
N = 10;
widths=2^N;
index = linspace(0,pi*2,depth);
sin_value = cos(index);
%sin_value = sin_value * (depth/2 -1);
sin_value = sin_value * (depth);
sin_value = fix((sin_value)+0.5);
%plot(sin_value);
number = [0:depth];
fid=fopen('Sine_ROM.mif','w+');
fprintf(fid,'depth=%d;
        n',depth);
fprintf(fid,'width=%d;
        n',widths);
fprintf(fid,'address_radix=UNS;
        n');
fprintf(fid,'data_radix = DEC;
        n');
fprintf(fid,'Content Begin\n');
for i = 1 : depth
    fprintf(fid, '%d\t%d;\n', number(i), sin_value(i));
end
fprintf(fid,'end;');
fclose(fid);

Vga_led.sv

module VGA_LED_Emulator(
    input logic clk50, reset,
input logic [7:0] hex0, hex1, hex2, hex3, hex4, hex5, hex6, hex7, hex8, hex9, hex10, hex11, hex12, hex13, hex14, hex15, hex16, hex17, hex18, hex19, hex20, hex21, hex22, hex23, hex24, hex25, hex26, hex27, hex28, hex29, hex30, hex31, hex32, hex33, hex34, hex35, hex36, hex37, hex38, hex39, hex40, hex41, hex42, hex43, hex44, hex45, hex46, hex47, hex48, hex49, hex50, hex51, hex52, hex53, hex54, hex55, hex56, hex57, hex58, hex59,
output logic [7:0] VGA_R, VGA_G, VGA_B,
output logic VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);

/*@* 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle *@

* HCOUNT 1599 0 1279 1599 0
* _______________  | Video | _______________  | Video
* *
* |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
* _______________  |________ |
* |____| VGA_HS |____|
*/

parameter radius_square = 20'd 1024;

parameter HACTIVE = 11'd 1280,
    HFRONT_PORCH = 11'd 32,
    HSYNC = 11'd 192,
    HBACK_PORCH = 11'd 96,
    HTOTAL = HACTIVE + HFRONT_PORCH + HSYNC + HBACK_PORCH; //1600

parameter VACTIVE = 10'd 480,
    VFRONT_PORCH = 10'd 10,
    VSYNC = 10'd 2,
    VBACK_PORCH = 10'd 33,
    VTOTAL = VACTIVE + VFRONT_PORCH + VSYNC + VBACK_PORCH; //525

logic [10:0] hcount; // Horizontal counter
logic endOfLine;
always_ff @(posedge clk50 or posedge reset)
if (reset)          hcount <= 0;
else if (endOfLine) hcount <= 0;
else              hcount <= hcount + 11'd 1;

assign endOfLine = hcount == HTOTAL - 1;

// Vertical counter
logic [9:0] vcount;
logic[9:0] vcount_plus1;
logic endOfField;

always_ff @(posedge clk50 or posedge reset)
if (reset)          vcount <= 0;
else if (endOfLine)
                  if (endOfField)   vcount <= 0;
                  else              vcount <= vcount + 10'd 1;
else              vcount <= vcount + 10'd 1;

assign endOfField = vcount == VTOTAL - 1;

// Horizontal sync: from 0x520 to 0x57F
// 101 0010 0000 to 101 0111 1111
assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

assign VGA_SYNC_n = 1; // For adding sync to video signals; not used for VGA

// Horizontal active: 0 to 1279     Vertical active: 0 to 479
// 101 0000 0000 1280
// 01 1110 0000 480
// 110 0011 111 524
                   !( vcount[9] | (vcount[8:5] == 4'b1111) );

assign VGA_CLK = hcount[0]; // 25 MHz clock: pixel latched on rising edge

////////////////////////////////////////////////////////////////////////////////
/* added by cong */
/*      Framebuffer      */

/ // Space information //

```verilog
/////////////////////////////////////////////////////////////
logic     inEdge;
logic     insurface;
logic     inBall_1;
logic     inBall_2;
logic     inBall_3;
logic     inBall_4;
logic     inHole;
logic     inEndPoint;
logic     shelter;
logic [10:0]  hcount_s;
logic     hitSurface_s1;
logic     hitSurface_s2;
logic     hitSurface_s3;
logic     hitSurface_s4;
logic     hitSurface_s5;
logic     hitEdge_s1;
logic     hitEdge_s2;
logic     hitEdge_s3;
logic     hitEdge_s4;
logic     hitEdge_s5;
logic     hitSurface_b;

/ /////////////////////////////////////////////////////////////
/*      Framebuffer      */

/ /////////////////////////////////////////////////////////////
logic     fb_odd_wren;
logic[127:0]  fb_odd_out;
integer     fb_odd_waddr;
integer     fb_odd_raddr;
integer     fb_odd_addr;
logic     fb_even_wren;
logic[127:0]  fb_even_out;
integer     fb_even_waddr;
integer     fb_even_raddr;
integer     fb_even_addr;

/ /////////////////////////////////////////////////////////////
// Space information //

/ /////////////////////////////////////////////////////////////
integer     x;
```
integer y;
integer z;
integer x_r1;
integer y_r1;
integer z_r1;
integer x_r2;
integer y_r2;
integer z_r2;
integer x0;

integer sin__x;
integer cos__x;
integer sin__y;
integer cos__y;

integer scan_count;
integer vcount_integer;
integer hcount_s_pack;
integer hcount_s_surface;
integer hcount_s_edge;
integer hcount_s_ball_1;
integer hcount_s_ball_2;
integer hcount_s_ball_3;
integer hcount_s_ball_4;
integer hcount_s_hole;
integer hcount_s_endPoint;

reg [15:0][7:0] plane;
logic[127:0] plane_r;
reg [3:0] section_sel = 4’b0;

///////////////////////////
//  object information  //</br>
///////////////////////////

//  square one
integer x1_s1;
integer x2_s1;
integer y1_s1;
integer y2_s1;
integer z1_s1;
integer z2_s1;

//  square two
integer  x1_s2;
integer  x2_s2;
integer  y1_s2;
integer  y2_s2;
integer  z1_s2;
integer  z2_s2;

//  square three
integer  x1_s3;
integer  x2_s3;
integer  y1_s3;
integer  y2_s3;
integer  z1_s3;
integer  z2_s3;

//  square four
integer  x1_s4;
integer  x2_s4;
integer  y1_s4;
integer  y2_s4;
integer  z1_s4;
integer  z2_s4;

//  square five
integer  x1_s5;
integer  x2_s5;
integer  y1_s5;
integer  y2_s5;
integer  z1_s5;
integer  z2_s5;

//  end point
integer  x_ep;
integer  x1_ep;
integer  x2_ep;
integer  y_ep;
integer  z_ep;
integer  z1_ep;
integer  z2_ep;

//  ball one
integer  x_b1;
integer  y_b1;
integer  z_b1;
integer  x1_b1;
integer  y1_b1;
integer  z1_b1;
integer  x2_b1;
integer  y2_b1;
integer  z2_b1;

// ball two
integer  x_b2;
integer  y_b2;
integer  z_b2;
integer  x1_b2;
integer  y1_b2;
integer  z1_b2;
integer  x2_b2;
integer  y2_b2;
integer  z2_b2;

// ball three
integer  x_b3;
integer  y_b3;
integer  z_b3;
integer  x1_b3;
integer  y1_b3;
integer  z1_b3;
integer  x2_b3;
integer  y2_b3;
integer  z2_b3;

// ball four
integer  x_b4;
integer  y_b4;
integer  z_b4;
integer  x1_b4;
integer  y1_b4;
integer  z1_b4;
integer  x2_b4;
integer  y2_b4;
integer  z2_b4;

///////////////////////////////////////
/*ball select information*/
///////////////////////////////////////
reg[1:0]   ball_select;
integer x1_b;
integer y1_b;
integer z1_b;
integer x2_b;
integer y2_b;
integer z2_b;

/////////////////////
/*state information*/
/////////////////////
logic start = 1'b0;
logic gameover;

//////////////////////
/* set object value */
//  end point
assign x_ep = {{24{hex7[7]}}, hex7};
assign y_ep = {{24{hex8[7]}}, hex8};
assign z_ep = {{24{hex9[7]}}, hex9};
assign x1_ep = x_ep - 5;
assign x2_ep = x_ep + 5;
assign z1_ep = z_ep - 5;
assign z2_ep = z_ep + 5;

//  square one
assign x1_s1 = {{24{hex10[7]}}, hex10};
assign x2_s1 = {{24{hex11[7]}}, hex11};
assign y1_s1 = {{24{hex12[7]}}, hex12};
assign y2_s1 = {{24{hex13[7]}}, hex13};
assign z1_s1 = {{24{hex14[7]}}, hex14};
assign z2_s1 = {{24{hex15[7]}}, hex15};

//  square two
assign x1_s2 = {{24{hex16[7]}}, hex16};
assign x2_s2 = {{24{hex17[7]}}, hex17};
assign y1_s2 = {{24{hex18[7]}}, hex18};
assign y2_s2 = {{24{hex19[7]}}, hex19};
assign z1_s2 = {{24{hex20[7]}}, hex20};
assign z2_s2 = {{24{hex21[7]}}, hex21};

//  square three
assign x1_s3 = {{24{hex22[7]}}, hex22};
assign x2_s3 = {{24{hex23[7]}}, hex23};//50;
assign y1_s3 = {{24{hex24[7]}}, hex24};//0;
assign y2_s3 = {{24{hex25[7]}}, hex25};//10;
assign z1_s3 = {{24{hex26[7]}}, hex26};//10;
assign z2_s3 = {{24{hex27[7]}}, hex27};//50;

// square four
assign x1_s4 = {{24{hex28[7]}}, hex28};//20;
assign x2_s4 = {{24{hex29[7]}}, hex29};//50;
assign y1_s4 = {{24{hex30[7]}}, hex30};//20;
assign y2_s4 = {{24{hex31[7]}}, hex31};//10;
assign z1_s4 = {{24{hex32[7]}}, hex32};//40;
assign z2_s4 = {{24{hex33[7]}}, hex33};//50;

// square five
assign x1_s5 = {{24{hex34[7]}}, hex34};//50;
assign x2_s5 = {{24{hex35[7]}}, hex35};//20;
assign y1_s5 = {{24{hex36[7]}}, hex36};//20;
assign y2_s5 = {{24{hex37[7]}}, hex37};//10;
assign z1_s5 = {{24{hex38[7]}}, hex38};//0;
assign z2_s5 = {{24{hex39[7]}}, hex39};//10;

// ball one
assign x_b1 = {{24{hex40[7]}}, hex40};
assign y_b1 = {{24{hex41[7]}}, hex41};
assign z_b1 = {{24{hex42[7]}}, hex42};
assign x1_b1 = x_b1 - 4;
assign y1_b1 = y_b1 - 4;
assign z1_b1 = z_b1 - 4;
assign x2_b1 = x_b1 + 4;
assign y2_b1 = y_b1 + 4;
assign z2_b1 = z_b1 + 4;

// ball two
assign x_b2 = {{24{hex43[7]}}, hex43};
assign y_b2 = {{24{hex44[7]}}, hex44};
assign z_b2 = {{24{hex45[7]}}, hex45};
assign x1_b2 = x_b2 - 4;
assign y1_b2 = y_b2 - 4;
assign z1_b2 = z_b2 - 4;
assign x2_b2 = x_b2 + 4;
assign y2_b2 = y_b2 + 4;
assign z2_b2 = z_b2 + 4;
// ball three
assign x_b3 = {{24{hex46[7]}}, hex46};
assign y_b3 = {{24{hex47[7]}}, hex47};
assign z_b3 = {{24{hex48[7]}}, hex48};
assign x1_b3 = x_b3 - 4;
assign y1_b3 = y_b3 - 4;
assign z1_b3 = z_b3 - 4;
assign x2_b3 = x_b3 + 4;
assign y2_b3 = y_b3 + 4;
assign z2_b3 = z_b3 + 4;

// ball four
assign x_b4 = {{24{hex49[7]}}, hex49};
assign y_b4 = {{24{hex50[7]}}, hex50};
assign z_b4 = {{24{hex51[7]}}, hex51};
assign x1_b4 = x_b4 - 4;
assign y1_b4 = y_b4 - 4;
assign z1_b4 = z_b4 - 4;
assign x2_b4 = x_b4 + 4;
assign y2_b4 = y_b4 + 4;
assign z2_b4 = z_b4 + 4;

////////////////////////////////////////////
/*                Control Signal                   */
/////////////////////////////////////////////////////////
assign hcount_s = hcount - 11'd128;
assign hcount_s_pack = hcount_s[5:1] - 1'b1;
assign hcount_s_surface = {hcount_s_pack[4:1], 3'd0};
assign hcount_s_edge = {hcount_s_pack[4:1], 3'd1};
assign hcount_s_ball_1 = {hcount_s_pack[4:1], 3'd2};
assign hcount_s_ball_2 = {hcount_s_pack[4:1], 3'd3};
assign hcount_s_ball_3 = {hcount_s_pack[4:1], 3'd4};
assign hcount_s_ball_4 = {hcount_s_pack[4:1], 3'd5};
assign hcount_s_hole = {hcount_s_pack[4:1], 3'd6};
assign hcount_s_endPoint = {hcount_s_pack[4:1], 3'd7};

assign fb_odd_wren   = !vcount[1] && (hcount_s == 11'd290 || hcount_s == 11'd590 || hcount_s == 11'd890 || hcount_s == 11'd1190);
assign fb_even_wren   = vcount[1] && (hcount_s == 11'd290 || hcount_s == 11'd590 || hcount_s == 11'd890 || hcount_s == 11'd1190);
assign fb_odd_waddr   = {section_sel, vcount[8:2]};
assign fb_odd_raddr   = {hcount_s[9:6], vcount[8:2]};
assign fb_even_waddr  = {section_sel, vcount[8:2] + 7'd1};
assign fb_even_raddr = {hcount_s[9:6], vcount[8:2]};
assign fb_odd_addr   = fb_odd_wren  ? fb_odd_waddr : fb_odd_raddr;
assign fb_even_addr  = fb_even_wren ? fb_even_waddr : fb_even_raddr;

assign insurface   = vcount[1] ? fb_odd_out[hcount_s_surface] : fb_even_out[hcount_s_surface];
assign inEdge      = vcount[1] ? fb_odd_out[hcount_s_edge] : fb_even_out[hcount_s_edge];
assign inBall_1    = vcount[1] ? fb_odd_out[hcount_s_ball_1] : fb_even_out[hcount_s_ball_1];
assign inBall_2    = vcount[1] ? fb_odd_out[hcount_s_ball_2] : fb_even_out[hcount_s_ball_2];
assign inBall_3    = vcount[1] ? fb_odd_out[hcount_s_ball_3] : fb_even_out[hcount_s_ball_3];
assign inBall_4    = vcount[1] ? fb_odd_out[hcount_s_ball_4] : fb_even_out[hcount_s_ball_4];
assign inHole      = vcount[1] ? fb_odd_out[hcount_s_hole] : fb_even_out[hcount_s_hole];
assign inEndPoint  = vcount[1] ? fb_odd_out[hcount_s_endPoint] : fb_even_out[hcount_s_endPoint];

assign shelter   = hcount_s[10] || hcount_s[10:2] == 9'd0;

assign x1_b = ball_select == 0 ? x1_b1 :
    ball_select == 1 ? x1_b2 :
    ball_select == 2 ? x1_b3 :
    x1_b4;
assign y1_b = ball_select == 0 ? y1_b1 :
    ball_select == 1 ? y1_b2 :
    ball_select == 2 ? y1_b3 :
    y1_b4;
assign z1_b = ball_select == 0 ? z1_b1 :
    ball_select == 1 ? z1_b2 :
    ball_select == 2 ? z1_b3 :
    z1_b4;

assign x2_b = ball_select == 0 ? x2_b1 :
    ball_select == 1 ? x2_b2 :
    ball_select == 2 ? x2_b3 :
    x2_b4;
assign y2_b = ball_select == 0 ? y2_b1 :
    ball_select == 1 ? y2_b2 :
    ball_select == 2 ? y2_b3 :
    y2_b4;
assign z2_b = ball_select == 0 ? z2_b1 :
    ball_select == 1 ? z2_b2 :
ball_select == 2 ? z2_b3 : 
z2_b4;

/*----------------------------------------------------------------------------*/
/*                Buffer Mapping                  */
/*----------------------------------------------------------------------------*/

genvar i,j;
generate
for(j=0; j<16; j=j+1)
begin:plane_buffer_mapping_loop1
for(i=0; i<8; i=i+1)
begin: plane_buffer_mapping_loop2
assign plane_r[8*j + i] = plane[j][i];
end
end
endgenerate

/*----------------------------------------------------------------------------*/
/*                Function & Task                  */
/*----------------------------------------------------------------------------*/

function hitSurface;
  input integer x_r, y_r, z_r, x1_s, x2_s, y1_s, y2_s, z1_s, z2_s;
  begin
    hitSurface = (x_r == x1_s || x_r == x2_s) && y_r >= y1_s && y_r <= y2_s &&
    z_r >= z1_s && z_r <= z2_s ||
    (y_r == y1_s || y_r == y2_s) && z_r >= z1_s && z_r <= z2_s &&
    x_r >= x1_s && x_r <= x2_s ||
    (z_r == z1_s || z_r == z2_s) && x_r >= x1_s && x_r <= x2_s &&
    y_r >= y1_s && y_r <= y2_s;
  end
endfunction

function hitEdge;
  input integer x_r, y_r, z_r, x1_s, x2_s, y1_s, y2_s, z1_s, z2_s;
  begin
    hitEdge = (x_r == x1_s || x_r == x2_s) && (y_r == y1_s || y_r == y2_s) &&
    z_r >= z1_s && z_r <= z2_s ||
    (y_r == y1_s || y_r == y2_s) && (z_r == z1_s || z_r == z2_s) &&
    x_r >= x1_s && x_r <= x2_s ||
    (z_r == z1_s || z_r == z2_s) && (x_r == x1_s || x_r == x2_s) &&
    y_r >= y1_s && y_r <= y2_s;
  end
endfunction

/*----------------------------------------------------------------------------*/
always_ff @(posedge clk50) begin
  // ball select
  ball_select = ball_select + 2'b1;

  start <= ((hex4[0] == 1'b1) && gameover);

  // initial
  if (hcount_s == 11'd0 || hcount_s == 11'd300 || hcount_s == 11'd600 || hcount_s == 11'd900) begin
    scan_count <= 0;
    section_sel[2:0] <= section_sel[2:0] + 3'b1;
    for (x = 0; x < 16; x++) plane[x] <= 8'b0;
  end

  if (endOfLine && endOfField) begin
    section_sel[3] <= !section_sel[3];
  end

  if (scan_count != 256) begin
    scan_count <= scan_count + 1;
    for (x = 0; x < 16; x++) begin
      x0 = {24'b0, section_sel, x[3:0]} - 128;
      x_r1 = x0;
      y_r1 = ((cos__x*y + sin__x*z)>>>12);
      z_r1 = ((-sin__x*y + cos__x*z)>>>12);
      x_r2 = ((-sin__y*z_r1 + cos__y*x_r1)>>>12);
      y_r2 = y_r1;
      z_r2 = ((cos__y*z_r1 + sin__y*x_r1)>>>12);
      if (hitSurface_b = hitSurface(x_r2, y_r2, z_r2, x1_b, x2_b, y1_b, y2_b, z1_b, z2_b));
      if (hitSurface_s1 = hitSurface(x_r2, y_r2, z_r2, x1_s1, x2_s1, y1_s1, y2_s1, z1_s1, z2_s1));
      if (hitSurface_s2 = hitSurface(x_r2, y_r2, z_r2, x1_s2, x2_s2, y1_s2, y2_s2, z1_s2, z2_s2));
      if (hitSurface_s3 = hitSurface(x_r2, y_r2, z_r2, x1_s3, x2_s3, y1_s3, y2_s3, z1_s3, z2_s3));
      if (hitSurface_s4 = hitSurface(x_r2, y_r2, z_r2, x1_s4, x2_s4, y1_s4, y2_s4, z1_s4, z2_s4));
  end
end
*/
hitSurface_s5 = hitSurface(x_r2, y_r2, z_r2, x1_s5, x2_s5, y1_s5, y2_s5, z1_s5, z2_s5);

hitEdge_s1 = hitEdge(x_r2, y_r2, z_r2, x1_s1, x2_s1, y1_s1, y2_s1, z1_s1, z2_s1);

hitEdge_s2 = hitEdge(x_r2, y_r2, z_r2, x1_s2, x2_s2, y1_s2, y2_s2, z1_s2, z2_s2);

hitEdge_s3 = hitEdge(x_r2, y_r2, z_r2, x1_s3, x2_s3, y1_s3, y2_s3, z1_s3, z2_s3);

hitEdge_s4 = hitEdge(x_r2, y_r2, z_r2, x1_s4, x2_s4, y1_s4, y2_s4, z1_s4, z2_s4);

hitEdge_s5 = hitEdge(x_r2, y_r2, z_r2, x1_s5, x2_s5, y1_s5, y2_s5, z1_s5, z2_s5);

if (plane[x] == 8'b0) plane[x][0] <= hitSurface_s1 || hitSurface_s2 || hitSurface_s3 || hitSurface_s4 || hitSurface_s5;

if (plane[x] == 8'b0) plane[x][1] <= hitEdge_s1 || hitEdge_s2 || hitEdge_s3 || hitEdge_s4 || hitEdge_s5; // || ball_edge;


if (plane[x] == 8'b0) plane[x][7] <= x_r2 >= x1_ep && x_r2 <= x2_ep && y_r2 == y_ep && z_r2 == z1_ep && z_r2 <= z2_ep;

if (plane[x] == 8'b0 && ball_select == 2'd0) plane[x][2] <= hitSurface_b;

if (plane[x] == 8'b0 && ball_select == 2'd1) plane[x][3] <= hitSurface_b;

if (plane[x] == 8'b0 && ball_select == 2'd2) plane[x][4] <= hitSurface_b;

if (plane[x] == 8'b0 && ball_select == 2'd3) plane[x][5] <= hitSurface_b;

end

end
sin(x).sin(address(address_sinx), .clock(clk50), .q(sin_x));
cos(x).cos(address(address_cosx), .clock(clk50), .q(cos_x));
sin(y).sin(address(address_siny), .clock(clk50), .q(sin_y));
cos(y).cos(address(address_cosy), .clock(clk50), .q(cos_y));

FrameBuffer fb_odd(.address(fb_odd_addr),
                    .clock(clk50),
                    .data(plane_r),
                    .wren(fb_odd_wren),
                    .q(fb_odd_out));

FrameBuffer fb_even(.address(fb_even_addr),
                     .clock(clk50),
                     .data(plane_r),
                     .wren(fb_even_wren),
                     .q(fb_even_out));

logic [11:0] address_sinx;
logic [11:0] address_cosx;
logic [11:0] address_siny;
logic [11:0] address_cosy;

logic [29:0] gameover_counter = 30'b0;
logic [1:0] state, next;
logic flash = 1'b0;
logic [5:0] flash_counter = 6'b0;

always_ff @(posedge clk50)
if(gameover)
   gameover_counter <= gameover_counter + 1'b1;
else
   gameover_counter <= 1'b0;

assign address_sinx = gameover ? gameover_counter[29:18] : {hex3[3:0],hex2};
assign address_cosx = gameover ? gameover_counter[29:18] : {hex3[3:0],hex2};
assign address_siny = gameover ? gameover_counter[29:18] : {hex1[3:0],hex0};
assign address_cosy = gameover ? gameover_counter[29:18] : {hex1[3:0],hex0};
assign gameover = hex6[0];
always_comb begin
    vcount_integer = vcount[9:1];
y = vcount_integer - 128;
z = scan_count - 128;
    {VGA_R, VGA_G, VGA_B} = flash ? {8'hff, 8'hff, 8'hff} : {8'h0, 8'h0, 8'h0}; // black

case (gameover)
    1'b0: begin
        if (insurface) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff}; // white
        if (inHole) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'h66}; // golden

        if (inEndPoint) {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0}; // black
        if (inEdge) {VGA_R, VGA_G, VGA_B} = {8'h66, 8'h66, 8'h66}; // pink
        if (inBall_1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h30, 8'h30}; // red
        if (inBall_3) {VGA_R, VGA_G, VGA_B} = {8'h56, 8'hca, 8'h32}; // green

        if (inBall_4) {VGA_R, VGA_G, VGA_B} = {8'h66, 8'h66, 8'h66}; // blue
        if (shelter) {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0}; // black
    end

    1'b1: begin
        if (insurface) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, gameover_counter[25:18]}; // white
        if (inEdge) {VGA_R, VGA_G, VGA_B} = {gameover_counter[25:18], 8'h0}; // black
        if (inBall_1) {VGA_R, VGA_G, VGA_B} = {8'hff, gameover_counter[25:18], 8'h0}; // pink

        if (inBall_2) {VGA_R, VGA_G, VGA_B} = {gameover_counter[25:18], 8'hca, gameover_counter[25:18]}; // green
        if (inBall_3) {VGA_R, VGA_G, VGA_B} = {8'hff, gameover_counter[25:18], 8'hdd}; // yellow
        if (inBall_4) {VGA_R, VGA_G, VGA_B} = {gameover_counter[25:18], gameover_counter[25:18], 8'hff}; // blue
        if (shelter) {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0}; // black
    end
endcase
end
endmodule