MASTER ELECTRONIC DESIGN

Homework 5(Cilindro)

Miguel Tlapa Juárez

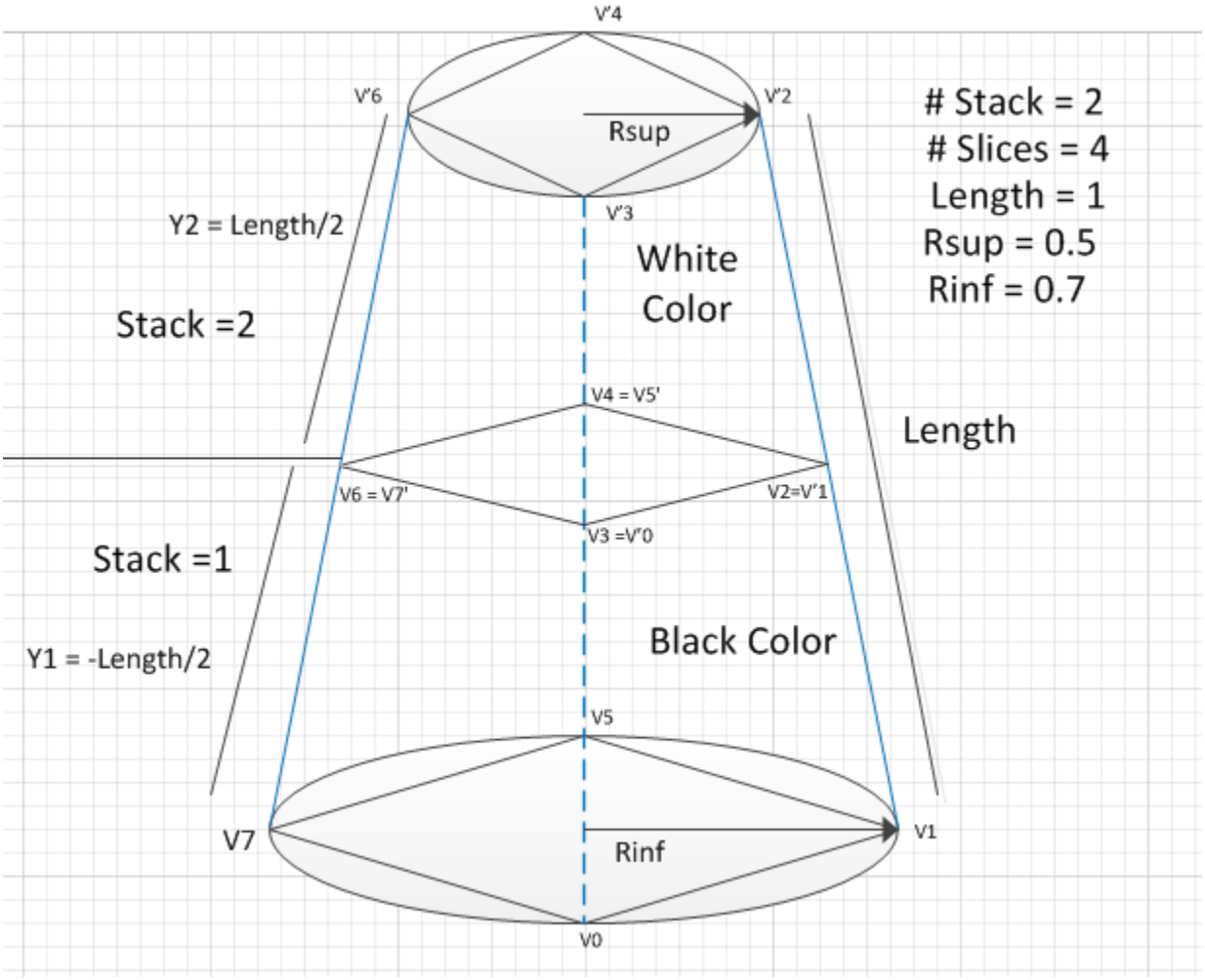
17/03/2014

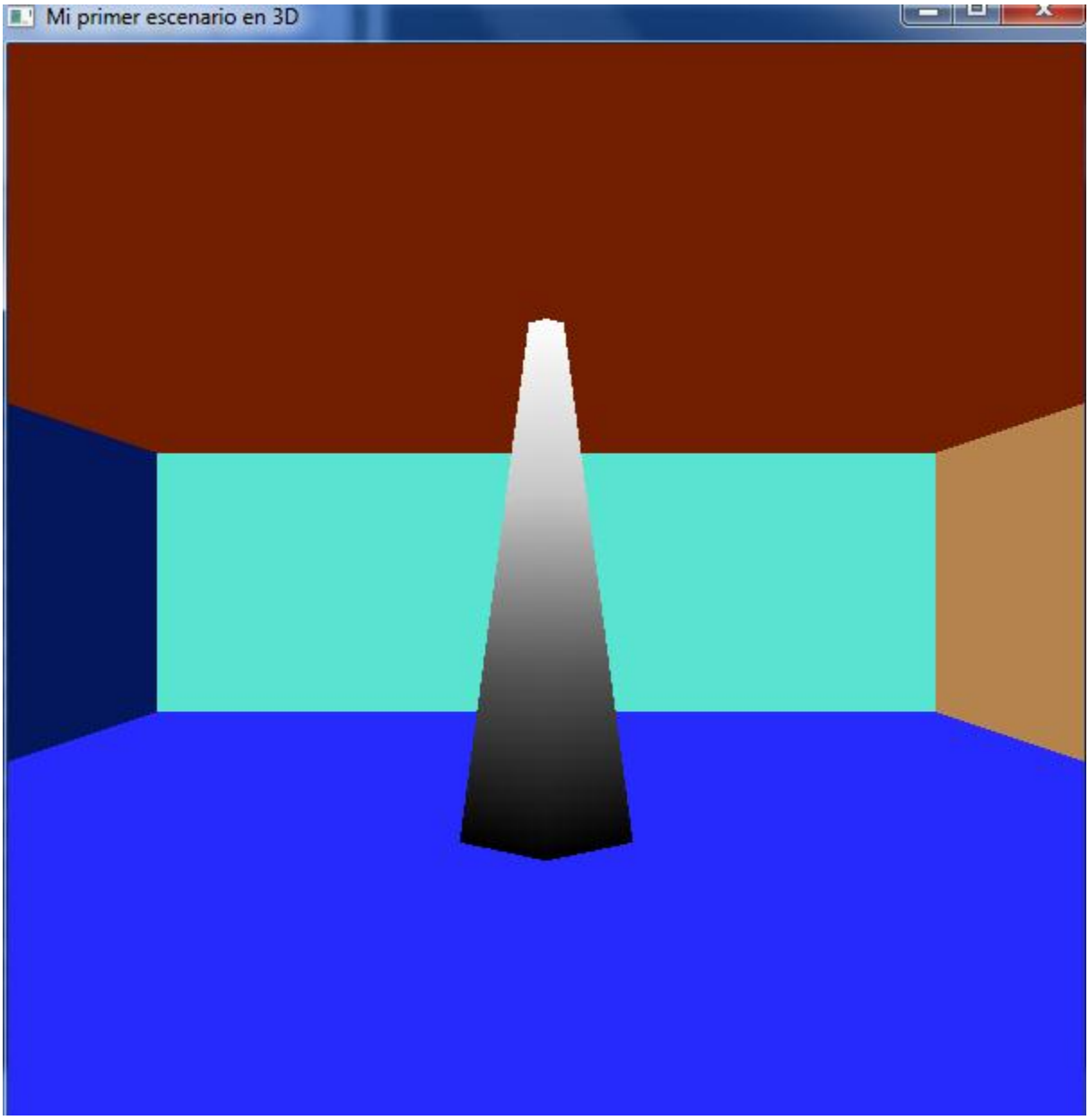
This document describes the system architecture and design about the body controller module, it's have block diagram and flowchart to describe software and hardware architecture.

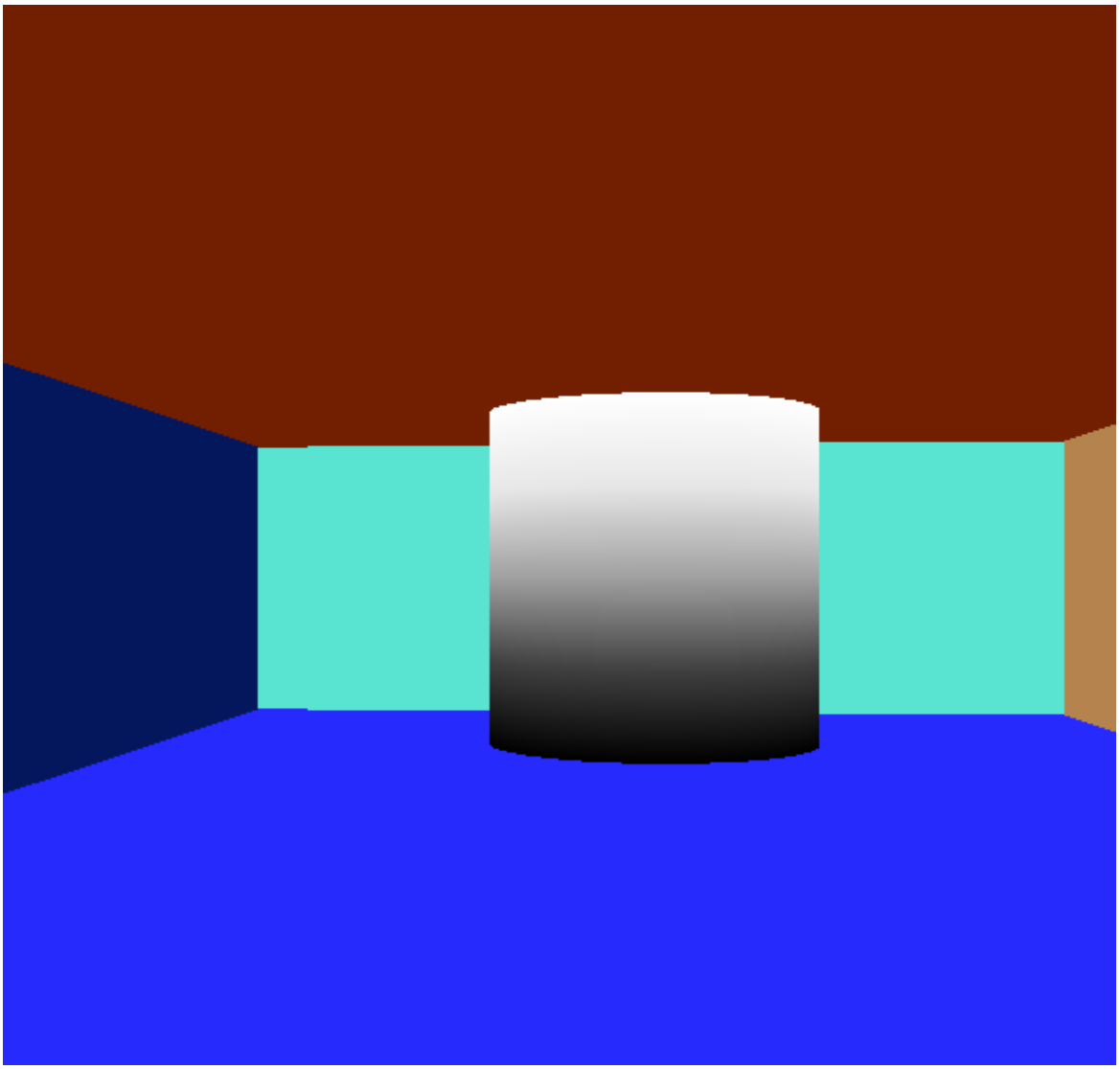
Revision History

Date	Revision Number	Author/Editor	Modifications
January 2014	0.1	Miguel Tlapa	Created file

Disclaimers


V'4
V'6
V'2
Rsup
# Stack = 2
# Slices = 4
Length = 1
Rsup = 0.5
Rinf = 0.7
V'3
Y2 = Length/2
White
Color
Stack =2
V4 = V5'
Length
V6 = V7'
V2=V'1
V3 =V'0
Stack =1
Black Color
Y1 = -Length/2
V5
V7
V1
Rinf
V0


Mi primer escenario en 3D



CYLINDER.CPP

```
/*
 * Cylinder.cpp
 *
 *  Created on: Mar 24, 2014
 *      Author: mtlapa
```

```
 */
#include <GL/glew.h>
#include "Cylinder.h"
#include <cstdlib>
#include <cmath>
#include <iostream>
using namespace std;

namespace cyl {
Cylinder::Cylinder(float length, float radius_sup, float radius_inf, int slices, int stacks){
	this->slices = slices;
	this->stacks = stacks;

	int size_positions = 2*3*(this->slices+1)*(this->stacks);
	this->positions    = new float[size_positions];
	int size_colors    = 2*3*(this->slices+1)*(this->stacks);
	this->colors       = new float[size_colors];
	int size_indexes   = ((this->slices + 1)*2*this->stacks) + (this->stacks-1);
	this->indexes      = new GLushort[size_indexes];

	reshape(length, radius_sup, radius_inf);
	setColors(0 , 0, 0, 1, 1, 1);

	int offset = 0;
	int restart = 1+(this->slices+1)*2;
	for (int i = 0; i < size_indexes; i++) {
```

```
			if((i+1)%restart == 0){
				this->indexes[i] = 0xffff;
				offset++;
			}
			else this->indexes[i] = i-offset;
		}

		for (int var = 0; var < size_indexes; var++) {
			if(indexes[var] == 0xffff) cout << indexes[var] << endl;
			else cout << indexes[var] << " ";
		}
}


void Cylinder::init(GLuint posLoc, GLuint colorLoc) {
		glGenVertexArrays(1, &vertexArrayId);
		glBindVertexArray(vertexArrayId);
		glGenBuffers(3, bufferId);

		int size_positions = 2*3*(this->slices+1)*(this->stacks);
		int size_colors    = 2*3*(this->slices+1)*(this->stacks);
		int size_indexes   = ((this->slices + 1)*2*this->stacks) + (this->stacks-1);

		glBindBuffer(GL_ARRAY_BUFFER, bufferId[0]);
		glBufferData(GL_ARRAY_BUFFER, sizeof(this->positions)*size_positions, this->positions, GL_DYNAMIC_DRAW);
		glVertexAttribPointer(posLoc, 3, GL_FLOAT, 0, 0, 0);
```

```
	glEnableVertexAttribArray(posLoc);


	glBindBuffer(GL_ARRAY_BUFFER, bufferId[1]);
	glBufferData(GL_ARRAY_BUFFER, sizeof(this->colors)*size_colors, this->colors, GL_DYNAMIC_DRAW);
	glVertexAttribPointer(colorLoc, 3, GL_FLOAT, 0, 0, 0);
	glEnableVertexAttribArray(colorLoc);


	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, bufferId[2]);
	glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(this->indexes)*size_indexes, this->indexes,
GL_STATIC_DRAW);
}


void Cylinder::reshape(float length, float radius1, float radius2)
{
	int size_stack = 2*3*(this->slices+1);
	int size_positions = size_stack*(this->stacks);


	float stack  = length/this->stacks;
	float y1 = -length/2;
	float y2 = (-length/2)+stack;


	float radians     = 0 ;
	float inc_radians = 2.0 * M_PI /this->slices;


	float inc_radius = (radius1 - radius2)/this->stacks;
	float rad1 = radius1;
```

```
	float rad2 = radius1 - inc_radius;

	for(int i = 0; i < size_positions; i+=6) {
		if(i%size_stack == 0 && i != 0){
			rad1 = rad2;
			rad2 -= inc_radius;
			y1 = y2;
			y2 = y1 + stack;

		}
		this->positions[i]     = rad1*cos(radians);   //Position X Superior
		this->positions[i + 1] = y1;      //Position Y Superior
		this->positions[i + 2] = rad1*sin(radians);  //Position Z Superior

		this->positions[i + 3] = rad2*cos(radians);  //Position X Inferior
		this->positions[i + 4] = y2;  //Position Y Inferior
		this->positions[i + 5] = rad2*sin(radians);  //Position Z Inferior
		radians += inc_radians;
	}

	for (int i = 0; i <= size_positions; i++)
	{
		if (i%3 ==0)cout << endl;
		cout << ' ' << positions[i];
	}
}
```

```
void Cylinder::setColors(float red1, float green1, float blue1, float red2, float green2, float blue2){

        int vertexColorMax = (this->slices+1) * 2 * 3 * this->stacks;
            int vertexByStack = (this->slices+1) * 2 * 3;
            float stack = 1.0;
            float weigth1 = 1.0 - (stack/(float) this->stacks);
            float weigth2 = stack/(float) this->stacks;



            float red, green, blue;
            red   = (red1*weigth1) + (red2*weigth2);
            green = (green1*weigth1) + (green2*weigth2);
            blue  = (blue1*weigth1) + (blue2*weigth2);



            for(int i = 0; i < vertexColorMax; i+=6) {
                    if( i%vertexByStack == 0 && i != 0){
                            stack += 1;
                            weigth1 = 1.0 - (stack/(float) this->stacks);
                            weigth2 = stack/(float) this->stacks;
                            red1 = red;
                            green1 = green;
                            blue1 = blue;
                            red   = (red1*weigth1) + (red2*weigth2);
                            green = (green1*weigth1) + (green2*weigth2);
```

```
                    blue  = (blue1*weigth1) + (blue2*weigth2);
                }
                this->colors[i]   = red1;
                this->colors[i+1] = green1;
                this->colors[i+2] = blue1;
                this->colors[i+3] = red;
                this->colors[i+4] = green;
                this->colors[i+5] = blue;
            }



            }

void Cylinder::draw(){
    GLboolean pr = glIsEnabled(GL_PRIMITIVE_RESTART);
    GLboolean cf = glIsEnabled(GL_CULL_FACE);

    glEnable(GL_PRIMITIVE_RESTART);
    glPrimitiveRestartIndex(0xFFFF);
    glEnable(GL_CULL_FACE);

    int size_indexes   = ((this->slices + 1)*2*this->stacks) + (this->stacks-1);

    glBindVertexArray(vertexArrayId);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, bufferId[2]);
    glDrawElements(GL_TRIANGLE_STRIP, size_indexes, GL_UNSIGNED_SHORT, 0);
```

```
	if(!pr) glDisable(GL_PRIMITIVE_RESTART);
	if(!cf) glDisable(GL_CULL_FACE);
}

Cylinder::~Cylinder() {
	delete[] this->positions;
	delete[] this->indexes;
	delete[] this->colors;
	glDeleteVertexArrays(1, &vertexArrayId);
}




} /* namespace cyl */
```

CYLYNDER.H

```
/*
 * Cylinder.h
 *
 *  Created on: Mar 24, 2014
 *      Author: mtlapa
 */

#ifndef CYLINDER_H_
```

```
#define CYLINDER_H_
#include <GL/glew.h>
namespace cyl {

class Cylinder {
private:
	int slices;
	int stacks;
   float *positions;
   float *colors;
   GLushort *indexes;
   GLuint vertexArrayId;
   GLuint bufferId[3];
public:

   Cylinder(float length, float radius_sup, float radius_inf, int slices, int stacks);
	virtual ~Cylinder();
   void init(GLuint posLoc, GLuint colorLoc);
   void draw();
   void reshape(float length, float radius1, float radius2);
   void setColors(float red1, float green1, float blue1, float red2, float green2, float blue2);



};
```

} /* namespace cyl */

#endif /* CYLINDER_H_ */

******* APLICACION****

```
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <cmath>
#include "Transforms.h"
#include "Utils.h"
#include "Box.h"
#include "Matrix4.h"
#include "Room.h"
#include "Cylinder.h"

using namespace CG;
using namespace std;
using namespace cyl;


Matrix4 modelMatrix12, projectionMatrix12,viewMatrix12;
GLuint programId12, vertexPositionLoc12, vertexColorLoc12, modelMatrixLoc12,
projectionMatrixLoc12,viewMatrixLoc12;
Box box1(3, 2, 1);
Room room1(30, 10, 70);
Cylinder cylinder(4, 2, 2, 32, 4);
int motionType =0;
float cameraX = 0;
float cameraZ = 0;
float cameraAngle = 0;
float rotationSpeed = 0.5;
float speed = 1;

void moveForward() {
	float radians = cameraAngle*M_PI/180;
	cameraX -= speed*sin(radians);
	cameraZ -= speed*cos(radians);

}

void moveBack() {
```

```
	float radians = cameraAngle*M_PI/180;
	cameraX += speed*sin(radians);
	cameraZ += speed*cos(radians);
}

void turnLeft() {
	cameraAngle -= rotationSpeed;
}

void turnRight() {
	cameraAngle += rotationSpeed;
}
void initShaders12() {
	GLuint vShader = Utils::compileShader("shaders/mvp_col_pos.vsh",
GL_VERTEX_SHADER);

	if(!Utils::shaderCompiled(vShader)) return;
//	cout << "hola" <<endl;
	GLuint fShader = Utils::compileShader("shaders/color.fsh",
GL_FRAGMENT_SHADER);
	if(!Utils::shaderCompiled(fShader)) return;

	programId12 = glCreateProgram();
	glAttachShader(programId12, vShader);
	glAttachShader(programId12, fShader);
	glLinkProgram(programId12);
//	cout << programId12 << endl;

	vertexPositionLoc12   = glGetAttribLocation(programId12, "vertexPosition");
	vertexColorLoc12      = glGetAttribLocation(programId12, "vertexColor");
	modelMatrixLoc12      = glGetUniformLocation(programId12, "modelMatrix");
	projectionMatrixLoc12 = glGetUniformLocation(programId12,
"projectionMatrix");
	viewMatrixLoc12       = glGetUniformLocation(programId12, "viewMatrix");
}

void initModels12() {
//	box1.init(vertexPositionLoc12, vertexColorLoc12);
//	box1.setPosition(0, 5, -25);
//	box1.yAngle = 0;
	room1.init(vertexPositionLoc12, vertexColorLoc12);
	cylinder.init(vertexPositionLoc12, vertexColorLoc12);
}

void displayFunc12() {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
      glUseProgram(programId12);
      glUniformMatrix4fv(projectionMatrixLoc12, 1, true,
projectionMatrix12.values);

      switch(motionType){
            case 1: moveForward(); break;
            case 2: moveBack(); break;
            case 3: turnLeft(); break;
            case 4: turnRight(); break;
      }

      viewMatrix12.setIdentity();
      rotateY(viewMatrix12,-cameraAngle);
      translate(viewMatrix12,-cameraX,0,-cameraZ);
      glUniformMatrix4fv(viewMatrixLoc12, 1, true, viewMatrix12.values);

      modelMatrix12.setIdentity();
      glUniformMatrix4fv(modelMatrixLoc12, 1, true, modelMatrix12.values);
      room1.draw();

//
//    translate(modelMatrix12, box1.x, box1.y, box1.z);
//    rotateY(modelMatrix12, box1.yAngle += 0.5);
//    glUniformMatrix4fv(modelMatrixLoc12, 1, true, modelMatrix12.values);
//    box1.draw();


      glUniformMatrix4fv(modelMatrixLoc12, 1, true, modelMatrix12.values);
      cylinder.draw();

      glutSwapBuffers();
}

void reshapeFunc12(int w, int h) {
    if(h == 0) h = 1;
    glViewport(0, 0, w, h);
    float aspect = (float) w / h;
    projectionMatrix12.setPerspective(45, aspect, 0.1, 500);
}



void timerFunc12(int id) {
      glutTimerFunc(10, timerFunc12, id);
      glutPostRedisplay();
}
```

```
void keyReleasedFunc12(int key,int x, int y) {
      motionType = 0;

}

void keyPressedFunc12(int key, int x, int y) {
      switch(key)
      {
      case GLUT_KEY_UP: motionType = 1; break;
      case GLUT_KEY_DOWN: motionType = 2; break;
      case GLUT_KEY_RIGHT: motionType = 3; break;
      case GLUT_KEY_LEFT: motionType  = 4;


      }
//    cout << motionType << endl;
}

void exitFunc12(unsigned char key, int x, int y) {
    if (key == 27) {
        exit(0);
    }
 }

int main(int argc, char **argv) {
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize(600, 600);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Mi primer escenario en 3D");
    glutDisplayFunc(displayFunc12);
    glutReshapeFunc(reshapeFunc12);
    glutTimerFunc(10, timerFunc12, 1);
    glutKeyboardFunc(exitFunc12);
    glutSpecialFunc(keyPressedFunc12);
    glutSpecialUpFunc(keyReleasedFunc12);
    glewInit();
    glEnable(GL_DEPTH_TEST);
    initShaders12();
    initModels12();
    glClearColor(0.7, 0.8, 0.8, 1.0);
    glutMainLoop();
      return 0;
}
```