Graphical user interface, application

Description automatically generated![A picture containing table, indoor, plant

Description automatically generated]()

Juice Texture: ![Calendar

Description automatically generated with medium confidence]()Bamboo Texture: A picture containing text, grass, outdoor, green

Description automatically generated

**MAIN CODE:**

#include <stdlib.h> //include libraries

#include <GL/glut.h> //glut library

#include <math.h>

#include<stb/stb\_image.h>

#include"Texture.h"

#include"shaderClass.h"

int sbwin, dbwin;

int angle;

//Light

void

myinit(void)

{

GLfloat light\_ambient[] =

{ 0.3, 0.3, 0.3, 1.0 };

GLfloat light\_diffuse[] =

{ 6.0, 6.0, 6.0, 1.0 };

GLfloat light\_specular[] =

{ 1.0, 1.0, 1.0, 1.0 };

GLfloat light\_position[] =

{ -1.0, 1.0, 1.0, 0.0 };

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light\_ambient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light\_diffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, light\_specular);

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

glEnable(GL\_LIGHT0);

glDepthFunc(GL\_LESS);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_LIGHTING);

}

void renderScene(void) {

//camera and mouse

// angle of rotation for the camera direction

float angle = 0.0;

// actual vector representing the camera's direction

float lx = 0.0f, lz = -1.0f;

// XZ position of the camera

float x = 0.0f, z = 5.0f;

float fraction = 0.1f;

switch (key) {

case GLUT\_KEY\_W:

angle -= 0.01f;

lx = sin(angle);

lz = -cos(angle);

break;

case GLUT\_KEY\_A:

angle += 0.01f;

lx = sin(angle);

lz = -cos(angle);

break;

case GLUT\_KEY\_S:

x += lx \* fraction;

z += lz \* fraction;

break;

case GLUT\_KEY\_D:

x -= lx \* fraction;

z -= lz \* fraction;

break;

}

}

// Clear Color and Depth Buffers

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// Reset transformations

glLoadIdentity();

// Set the camera

gluLookAt(x, 1.0f, z,

x + lx, 1.0f, z + lz,

0.0f, 1.0f, 0.0f);

void

display(void)

{

//red vase

static GLfloat red[] = //color of torus (main part of red vase)

{ 0.8, 0.0, 0.0, 1.0 };

static GLfloat yellow[] =

{ 0.56, 0.56, 0.56, 1.0 }; //color of pyramid (base of red vase)

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glPushMatrix();

glRotatef(angle, 1.0, 0.0, 0.0);

glScalef(1.3, 1.3, 1.3);

glRotatef(20.0, 1.0, 0.0, 0.0); //torus placement

glPushMatrix();

glTranslatef(-0.75, 0.5, 0.0);

glRotatef(90.0, 1.0, 0.0, 0.0);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, red);

glutSolidTorus(0.56, 0.49, 10, 15); //first two numbers dictate size of torus

glPopMatrix();

glPushMatrix();

glTranslatef(-0.75, -0.5, 0.0);

glRotatef(270.0, 1.0, 0.0, 0.0);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, yellow);

glutSolidCone(1.0, 1.0, 40, 40); //dictates size of pyramid

glPopMatrix();

//plane (record table)

glBegin(GL\_QUADS);

/\* Floor \*/

glVertex3f(-1,-1,-1);

glVertex3f(1,-1,-1);

glVertex3f(1,-1,1);

glVertex3f(-1,-1,1);

//cube (juice box)

/\* Setup cube vertex data. \*/

v[0][0] = v[1][0] = v[2][0] = v[3][0] = -1;

v[4][0] = v[5][0] = v[6][0] = v[7][0] = 1;

v[0][1] = v[1][1] = v[4][1] = v[5][1] = -1;

v[2][1] = v[3][1] = v[6][1] = v[7][1] = 1;

v[0][2] = v[3][2] = v[4][2] = v[7][2] = 1;

v[1][2] = v[2][2] = v[5][2] = v[6][2] = -1;

//light

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light\_diffuse);

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

glEnable(GL\_LIGHT0);

glEnable(GL\_LIGHTING); glEnable(GL\_DEPTH\_TEST);

//projection

glMatrixMode(GL\_PROJECTION);

gluPerspective( 40.0, 1.0, 1.0, 10.0);

glMatrixMode(GL\_MODELVIEW);

gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0,

0.0, 1.0, 0.);

glTranslatef(0.0, 0.0, -1.0); //placement of cube

glRotatef(60, 1.0, 0.0, 0.0); //size of cube

glRotatef(-20, 0.0, 0.0, 1.0);

//cylinders (two bamboo stalks, straw, and yellow vase)

void gluCylinder(GLU quadric\* quad,

GLdouble base,

GLdouble top,

GLdouble height,

GLint slices,

GLint stacks);

void OnRender(float pHeight) {

glClearColor(1.0f, 0.0f, 0.0f, 1.0f); //clear, we will add texture

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

glLoadIdentity();

void gluCylinder(GLU quadric\* quad,

GLdouble base,

GLdouble top,

GLdouble height,

GLint slices,

GLint stacks);

void OnRender(float pHeight) {

glClearColor(1.0f, 0.0f, 0.0f, 1.0f); //clear, we will add texture

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

glLoadIdentity();

void gluCylinder(GLU quadric\* quad,

GLdouble base,

GLdouble top,

GLdouble height,

GLint slices,

GLint stacks);

void OnRender(float pHeight) {

glClearColor(0.0f, 0.0f, 1.0f, 1.0f); //yellow

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

glLoadIdentity();

void gluCylinder(GLU quadric\* quad,

GLdouble base,

GLdouble top,

GLdouble height,

GLint slices,

GLint stacks);

void OnRender(float pHeight) {

glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //white

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

glLoadIdentity();

gluCylinder(quadratic, 0.1f, 0.1f, pHeight, 32, 32);

glFlush();

}

glPopMatrix();

if (glutGetWindow() == sbwin) {

glFlush();

}

else {

glutSwapBuffers();

}

}

// Texture jpeg input for cylinder (bamboo)

Texture Tex("BAMBOOTEXTURE.png", GL\_TEXTURE\_2D, GL\_TEXTURE0, GL\_RGBA, GL\_UNSIGNED\_BYTE);

Tex.texUnit(shaderProgram, "tex0", 0);

// Texture jpeg input for cube (juicebox)

Texture Tex("JUICE.png", GL\_TEXTURE\_2D, GL\_TEXTURE0, GL\_RGBA, GL\_UNSIGNED\_BYTE);

Tex.texUnit(shaderProgram, "tex0", 0);

glutSwapBuffers();

}

void processSpecialKeys(int key, int xx, int yy) {

// Binds texture so that is appears in rendering

Tex.Bind();

void

reshapeOpenGLState(int w, int h)

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (w <= h)

glOrtho(-2.5, 2.5, -2.5 \* (GLfloat)h / (GLfloat)w,

2.5 \* (GLfloat)h / (GLfloat)w, -10.0, 10.0);

else

glOrtho(-2.5 \* (GLfloat)w / (GLfloat)h,

2.5 \* (GLfloat)w / (GLfloat)h, -2.5, 2.5, -10.0, 10.0);

glMatrixMode(GL\_MODELVIEW);

}

void

sbReshape(int w, int h)

{

glutSetWindow(sbwin);

glViewport(0, 0, w, h);

reshapeOpenGLState(w, h);

glutSetWindow(dbwin);

glutReshapeWindow(w, h);

}

void

dbReshape(int w, int h)

{

glViewport(0, 0, w, h);

reshapeOpenGLState(w, h);

}

void

rotation(void)

{

angle += 2;

angle = angle % 360;

glutPostRedisplay();

}

int animation = 0; //animation

void

main\_menu(int value)

{

switch (value) {

case 1:

//Smart toggle rotation ensures we switch to double buffered when

animating and single buffered when not animating

animation = 1 - animation; //Toggle

if (animation) {

glutIdleFunc(rotation);

glutSetWindow(sbwin);

glutSetWindowTitle("sb2db - double buffer mode");

glutSetWindow(dbwin);

glutShowWindow(); //Show the double buffered window

}

else {

glutIdleFunc(NULL);

glutSetWindow(sbwin);

glutSetWindowTitle("sb2db - single buffer mode");

glutSetWindow(dbwin);

glutHideWindow(); // Hide the double buffered window

}

break;

case 2:

glutSetWindow(dbwin);

glutHideWindow(); //Hide the double buffered window

glutSetWindow(sbwin);

glutSetWindowTitle("sb2db - single buffer mode");

break;

case 3:

glutSetWindow(sbwin);

glutSetWindowTitle("sb2db - double buffer mode");

glutSetWindow(dbwin);

glutShowWindow(); // Show the double buffered window

break;

case 4:

animation = 1 - animation; //Toggle

if (animation)

glutIdleFunc(rotation);

else

glutIdleFunc(NULL);

break;

case 666:

exit(0);

break;

}

}

void

visibility(int state)

{

static int sbvis = GLUT\_NOT\_VISIBLE, dbvis = GLUT\_NOT\_VISIBLE;

int eithervis;

if (glutGetWindow() == sbwin) {

sbvis = state;

}

else {

dbvis = state;

}

eithervis = (sbvis == GLUT\_VISIBLE) || (dbvis == GLUT\_VISIBLE);

if (eithervis) {

// Resume rotating idle callback if we become visible and

animation is enabled

if (animation) {

glutIdleFunc(rotation);

}

}

else {

// Disable animation when both windows are not visible

glutIdleFunc(NULL);

}

}

int

main(int argc, char\*\* argv)

{

glutInitWindowSize(500, 500);

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_DEPTH | GLUT\_SINGLE);

// The top window is single buffered

sbwin = glutCreateWindow(argv[0]);

glutReshapeFunc(sbReshape);

glutDisplayFunc(display);

glutVisibilityFunc(visibility);

myinit();

glutInitDisplayMode(GLUT\_RGB | GLUT\_DEPTH | GLUT\_DOUBLE);

dbwin = glutCreateSubWindow(glutGetWindow(),

0, 0, glutGet(GLUT\_WINDOW\_WIDTH), glutGet(GLUT\_WINDOW\_HEIGHT));

glutDisplayFunc(display);

glutReshapeFunc(dbReshape);

glutVisibilityFunc(visibility);

myinit();

//Initially hide the double buffered window to start in

single buffered mode.

glutHideWindow();

glutCreateMenu(main\_menu);

glutAddMenuEntry("Smart rotation toggle", 1);

glutAddMenuEntry("Single buffer", 2);

glutAddMenuEntry("Double buffer", 3);

glutAddMenuEntry("Toggle rotation", 4);

glutAddMenuEntry("Quit", 666);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutSetWindow(sbwin);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

return 0;

}