

Mihai Tuhari

mihai-razvan.tuhari@s.unibuc.ro

<https://github.com/mihaituhari/>

Grafica pe calculator

Proiect 1 (2D)

Dezvoltati un proiect 2D pentru tema aleasa.

Simulati o "depasire": o masina / un dreptunghi se deplaseaza uniform (prin translatie), un alt dreptunghi vine din spate (tot prin translatii/rotatii), la un moment dat intra in depasire, apoi trece in fata primului.



Demo

Pentru o lectura mai coerenta si simpla asupra documentatiei de mai jos, incepem cu o captura de ecran a proiectului.



Introducere

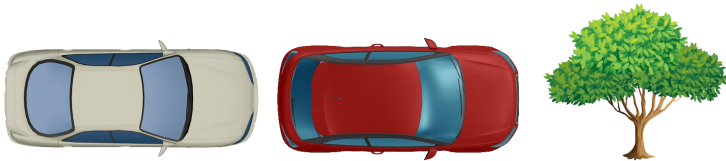
Am realizat o simulare a unei depasiri intre doua masini, folosind OpenGL si C++.

Programul principal este in fisierul [proiect1.cpp](#) si foloseste [libraria STB](#) pentru incarcarea texturilor.

Texturi

Pentru manipularea texturilor am ales STB in loc de SOIL pentru ca pe MacOS SOIL nu este compatibil.

Texturile se regasesc in folderul [textures](#) si sunt fisiere PNG cu fundal transparent:

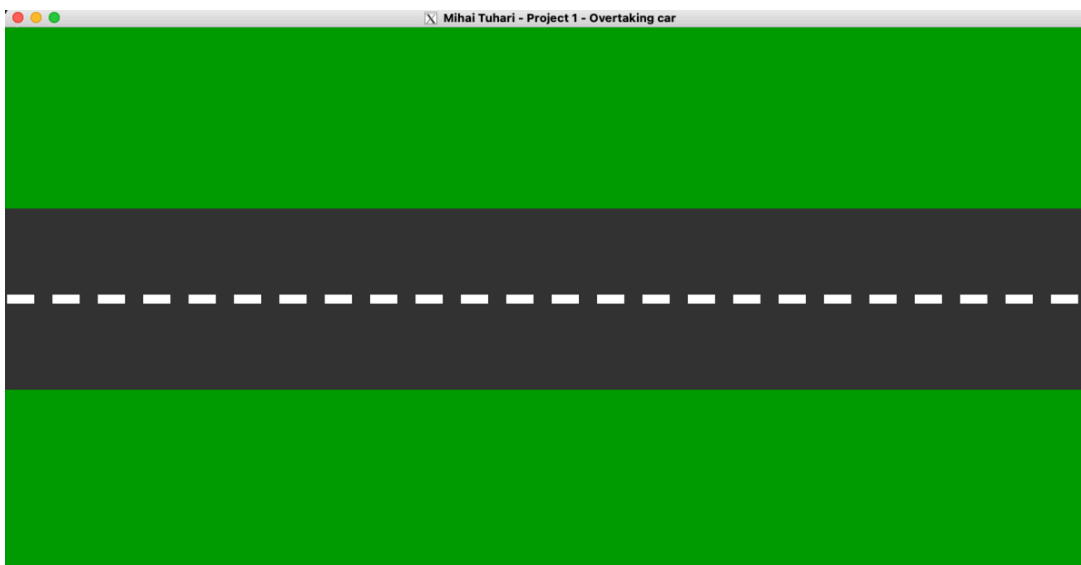


Conceptul


Animatia este creata din mai multe componente cu roluri diferite.


Scena statica

Scena este un dreptunghi care reprezinta drumul pe care se deplaseaza masinile. Este desenat cu culoarea verde iar asfaltul este gri.



Decor dinamic

 Pentru a simula miscarea, am adaugat un marcaj discontinuu pe mijlocul drumului prin functia `drawRoad()`. Aceasta genereaza dreptunghiuri albe cu o anumita distanta intre ele.

 In partea superioara sunt desenate copaci tot pentru a simula miscarea si pentru un plus de complexitate. Acestia sunt adaugati prin `drawTrees()` si sunt dreptunghiuri ce se repeta pe latimea ecranului si au proportii 39x32 peste care se aplica textura [tree.png](#) cu optiunile de blending `GL_SRC_ALPHA` si `GL_ONE_MINUS_SRC_ALPHA` pentru a afisa doar textura.



Masinile

Sunt adaugate 2 masini care se deplaseaza pe scena cu functia `drawCar()`.

- Masina 1 este cea inceata
- Masina 2 este cea rapida

Acestea sunt dreptunghiuri cu proportii 100x55 peste care se aplica texturile [car1.png](#) si [car2.png](#) cu optiuni de blending similare celor de la copaci.

Semnalizarea

Masina 2 are atasat si conceptul de "semnalizare" pentru a indica depasirea. 

Semnalizarea este un singur cerc galben (facut cu helperul `drawCircle()`) cu transparenta 70% ce este atasat de masina. Aceasta este o singura instanta si in functie de valoarea `blinkersOn` (`0=oprit`, `1=stanga`, `2=dreapta`) isi schimba pozitia pe axa Y fata de masina.

Programatic, am simulat o semnalizare apropiata de un comportament real:

- Semnalizarea de depasire (stanga) este pornita cand mai sunt 2 lungimi de masina pana la masina din fata
- Pe timpul depasirii, semnalizarea de depasire ramane pornita
- Dupa ce masina lenta a fost intrecuta cu o lungime de masina, este pusa semnalizare de revenire (dreapta)

Depasirea

In timpul schimbarii de banda, masina 2 schimba banda pe axa Y cu o anumita viteza si unghi de rotatie. La revenire, masina 2 se intoarce la banda initiala.

Aspecte tehnice

Design modular

Am abordat proiectul cu un design modular si am incercat sa folosesc cat mai multe functii pentru a separa logica.

Configurabilitate

Animatia este usor configurabila din variabilele globale definite la inceputul fisierului [proiect1.cpp](#).

Acolo regasiti variabile pentru:

- Globale (dimensiune extra, PI pentru cercuri)
- Texturi (cale fisiere, ID-uri, etc)
- Masini (pozitii initiala, viteze de deplasare, dimensiuni)
- Depasire (rotatie, rotatie maxima, buffer siguranta revenire din depasire)
- Drum (grosime marcaj, spatiere copaci)
- Scena (viteza animatie, interval repetare)

Transformari

Pentru a realiza animatia, am folosit urmatoarele transformari:

- Translatie
- Rotatie

Cand o masina ajunge la capatul ecranului, aceasta este repusa la inceputul drumului.

Limba

Intreg codul (cu tot cu comentarii), este scris in limba Engleza din motive de coerenta si simplitate, pentru a evita combinatia intre termeni in limba Romana si Engleza.

Video

Puteti vedea unvideo cu animatia in actiune la sfarsitul paginii

<https://github.com/mihaituhari/fmi/tree/main/qc/proiect1> ori direct [aici](#).

Codul sursa

Este inclus pe paginile urmatoare, dar il puteti regasi si pe contul meu de GitHub la adresa:

<https://github.com/mihaituhari/fmi/tree/main/qc/proiect1>

```

/**
 * Project 1 - Overtaking car
 *
 * @author Mihai Tuhari
 * @date November 2024
 */
#define STB_IMAGE_IMPLEMENTATION

#include <iostream>
#include <string>
#include <GL/freeglut.h>
#include "libs/stb_image.h"

// Globals
const int windowHeight = 1200;
const int windowHeight = 600;
const double PI = 3.141592653589793; // For circles

// Texture variables
GLuint textureIDCar1, textureIDCar2, textureTree;
const std::string texturePath = "/Volumes/mihai/dev/fmi/gc/proiect1/textures/";
int width, height, channels;

// Car positions and attributes
GLfloat car1X = 360, car1Y = 220;
GLfloat car2X = 30, car2Y = 220;
GLfloat carSpeed1 = 1, carSpeed2 = 3;
GLfloat carWidth = 110, carHeight = 55;

// Overtaking variables
GLfloat car2Rotation = 0;
const GLfloat maxRotation = 15;
GLfloat overtakeSafety = 20;

// Blinker variables
bool blinkerState = false; // On or off for interval
const GLint blinkerInterval = 500;
const GLfloat blinkerRadius = 10.0;
GLint blinkersOn = 0; // 0 = off, 1 = left blinker, 2 = right blinker

// Road elements
GLfloat roadDividerWidth = 5;
GLint treeSpacing = 120;

// Scene variables
GLfloat animationInterval = 16; // ~60 FPS (= 1000 / 60)
GLfloat sceneSpeed = 3.6;
GLfloat sceneOffset = 0;

// Flags
bool overtaking = false;
bool returningToLane = false;

// Circle drawing helper function
void drawCircle(GLfloat radius, GLint segments) {
    glBegin(GL_TRIANGLE_FAN);
    glVertex2f(0.0f, 0.0f);
    for (int i = 0; i <= segments; i++) {
        GLfloat angle = 2.0f * PI * i / segments;
        GLfloat x = radius * cos(angle);
        GLfloat y = radius * sin(angle);
    }
}

```

```

        glVertex2f(x, y);
    }
    glEnd();
}

// Load a texture from file
void loadTexture(const std::string &path, GLuint &textureID) {
    stbi_set_flip_vertically_on_load(1); // Flip the image vertically on load
    unsigned char *image = stbi_load(path.c_str(), &width, &height, &channels,
    STBI_rgb_alpha);

    if (!image) {
        std::cerr << "Failed to load texture: " << path << std::endl;
        exit(1);
    }

    glGenTextures(1, &textureID);
    glBindTexture(GL_TEXTURE_2D, textureID);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE,
    image);
    stbi_image_free(image);
}

// Draw the moving trees on the side of the road
void drawTrees() {
    int yOffset = 450;
    int rectWidth = 39 * 2.5;
    int rectHeight = 32 * 2.5;

    int numTrees = (windowWidth / treeSpacing) + 2; // Extra trees for smooth transition

    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, textureTree);

    for (int i = 0; i < numTrees; ++i) {
        GLfloat treeX = (i * treeSpacing) + fmod(sceneOffset, treeSpacing);

        glPushMatrix();
        glTranslatef(treeX, 0, 0);

        glBegin(GL_QUADS);
        glTexCoord2f(0.0, 0.0); // Bottom-left
        glVertex2i(-rectWidth / 2, yOffset);

        glTexCoord2f(1.0, 0.0); // Bottom-right
        glVertex2i(rectWidth / 2, yOffset);

        glTexCoord2f(1.0, 1.0); // Top-right
        glVertex2i(rectWidth / 2, yOffset + rectHeight);

        glTexCoord2f(0.0, 1.0); // Top-left
        glVertex2i(-rectWidth / 2, yOffset + rectHeight);
        glEnd();

        glPopMatrix();
    }
}

```

```

    glDisable(GL_TEXTURE_2D);
}

// Toggle blinker state
void toggleBlinker(int value) {
    if (blinkersOn) { // Only toggle if blinking is active
        blinkerState = !blinkerState;
    } else {
        blinkerState = false; // Ensure blinker is off when not needed
    }

    glutTimerFunc(blinkerInterval, toggleBlinker, 0);
}

// Draw asphalt road with moving dividers
void drawRoad() {
    // Asphalt
    glColor3f(0.2, 0.2, 0.2);
    glBegin(GL_QUADS);
    glVertex2f(0, 200);
    glVertex2f(windowWidth, 200);
    glVertex2f(windowWidth, 400);
    glVertex2f(0, 400);
    glEnd();

    // Road divider (moving vertical lines)
    glColor3f(1, 1, 1);

    for (GLfloat x = sceneOffset; x < windowWidth; x += 50) {
        glBegin(GL_QUADS);
        glVertex2f(x, 300 - roadDividerWidth); // Center of the road
        glVertex2f(x + 30, 300 - roadDividerWidth);
        glVertex2f(x + 30, 300 + roadDividerWidth);
        glVertex2f(x, 300 + roadDividerWidth);
        glEnd();
    }
}

// Draw a car
void drawCar(GLfloat x, GLfloat y, GLuint textureID, GLfloat rotationAngle = 0, bool
fasterCar = false) {
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, textureID);

    // Store the original matrix
    glPushMatrix();

    // Move to car position and apply rotation
    glTranslatef(x + carWidth / 2, y + carHeight / 2, 0);
    glRotatef(rotationAngle, 0.0f, 0.0f, 1.0f);

    // Draw car body
    glPushMatrix();
    glTranslatef(-carWidth / 2, -carHeight / 2, 0);
    glBegin(GL_QUADS);
    glTexCoord2f(0, 0);
    glVertex2f(0, 0);
    glTexCoord2f(1, 0);
    glVertex2f(carWidth, 0);
    glTexCoord2f(1, 1);
    glVertex2f(carWidth, carHeight);
    glTexCoord2f(0, 1);
    glVertex2f(0, carHeight);
}

```



```

glEnd();
glPopMatrix();

glDisable(GL_TEXTURE_2D);

// Draw blinker in the same transformed space (of the overtaking car)
if (fasterCar && blinkersOn && blinkerState) {
    GLfloat blinkerX, blinkerY;
    blinkerX = carWidth / 2 - blinkerRadius;
    blinkerY = blinkersOn == 1
        ? carHeight * 0.4 // Left blinker
        : -carHeight * 0.4; // Right blinker

    glTranslatef(blinkerX, blinkerY, 0);
    glColor4f(1.0, 1.0, 0.0, 0.7); // Yellow with transparency
    drawCircle(blinkerRadius / 2, 20);
}

glPopMatrix();
}

void init() {
    glClearColor(0, 0.6, 0, 1); // Green background, grass
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, windowHeight, 0.0, windowHeight);

    glutTimerFunc(blinkerInterval, toggleBlinker, 0);
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    drawRoad();

    // Enable blending for trees and cars
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    drawTrees();
    drawCar(car1X, car1Y, textureIDCar1, 0, false);
    drawCar(car2X, car2Y, textureIDCar2, car2Rotation, true);

    glDisable(GL_BLEND);
    glutSwapBuffers();
}

void update(int value) {
    // Move the screen
    sceneOffset -= sceneSpeed;

    // Move cars
    car1X += carSpeed1;
    car2X += carSpeed2;

    // Update blinker state based on car positions
    GLfloat distance = abs(car1X - car2X);
    if (car1X > car2X && distance <= carWidth * 3) {
        blinkersOn = 1; // Left blinker
    } else if (car2X > car1X && distance <= carWidth * 2) {
        blinkersOn = 2; // Right blinker
    } else {
        blinkersOn = 0; // No blinker
    }
}

```

```

}

// Handle the second car's overtaking behavior
if (!overtaking && !returningToLane) {
    // Start overtaking
    if (car1X > car2X && (car1X - car2X < (carWidth * 1.8))) {
        overtaking = true;
    }
} else if (overtaking) {
    // Move up and rotate the car for overtaking
    if (car2Y - car1Y < carHeight + overtakeSafety) {
        car2Y += 1;
        if (car2Rotation < maxRotation) {
            car2Rotation += 0.8;
        }
    } else {
        // In the overtaking position, parallel to the other car
        car2Rotation = 0;
    }

    // Overtaking completed. Will return to initial lane
    if (car2X - car1X > (carWidth)) {
        overtaking = false;
        returningToLane = true;
    }
} else if (returningToLane) {
    // Rotate and move down to the initial lane
    car2Y -= 1;
    if (car2Rotation > -maxRotation) {
        car2Rotation -= 0.8;
    }

    // Returned to initial lane
    if (car2Y <= 220) {
        car2Y = 220;
        car2Rotation = 0.0;
        returningToLane = false;
    }
}

// Loop cars when leaving the screen
if (car1X > windowWidth) {
    car1X = -carWidth;
}

if (car2X > windowWidth) {
    car2X = -carWidth;
}

glutPostRedisplay();
glutTimerFunc(animationInterval, update, 0);
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutCreateWindow("Mihai Tuhari - Project 1 - Overtaking car");

    init();

    loadTexture(texturePath + "car1.png", textureIDCar1);
    loadTexture(texturePath + "car2.png", textureIDCar2);

```

```
loadTexture(texturePath + "tree.png", textureTree);

glutDisplayFunc(display);
glutTimerFunc(animationInterval, update, 0);
glutMainLoop();

return 0;
}
```