Sapienza University of Rome

Interactive Computer Graphics

Sino: Haymaking challenge

Author:
Horpynchenko, Dmytro

Student Number:
1807584

Contents

1 Intro

Final project is a browser game with elements of 3D graphics.

1.1 Concept

The world of the game is a field with hay blocks and obstacles on it. A user is a driver of a vehicle that can move on the field in any direction and collect a hay. In case of collision with the obstacle, vehicle loose all its speed or accelerates in opposite direction in case of high speed. The aim of the game is to collect all hay blocks in defined time period. Amount of time, hay blocks and obstacles depends on the level of difficulty chosen by user.
The vehicle has 2 controls available:

- Front wheels steering (left and right) controlled with left and right arrow keys. After release of the keys vehicle's wheels return to their default positions of 0 deg.
- Accelerating (forward and backward) using up and down arrow keys accordingly. Release of the keys stop acceleration and leads to deceleration of the vehicle because of the wheel's friction.

Collecting action is done by colliding with hay block that lead to increase of the score and removing hay item from the field.

2 Implementation

2.1 Overview

2.1.1 World

The world of the game is represented as a simulation of real world with infinite grass field and a blue sky around.

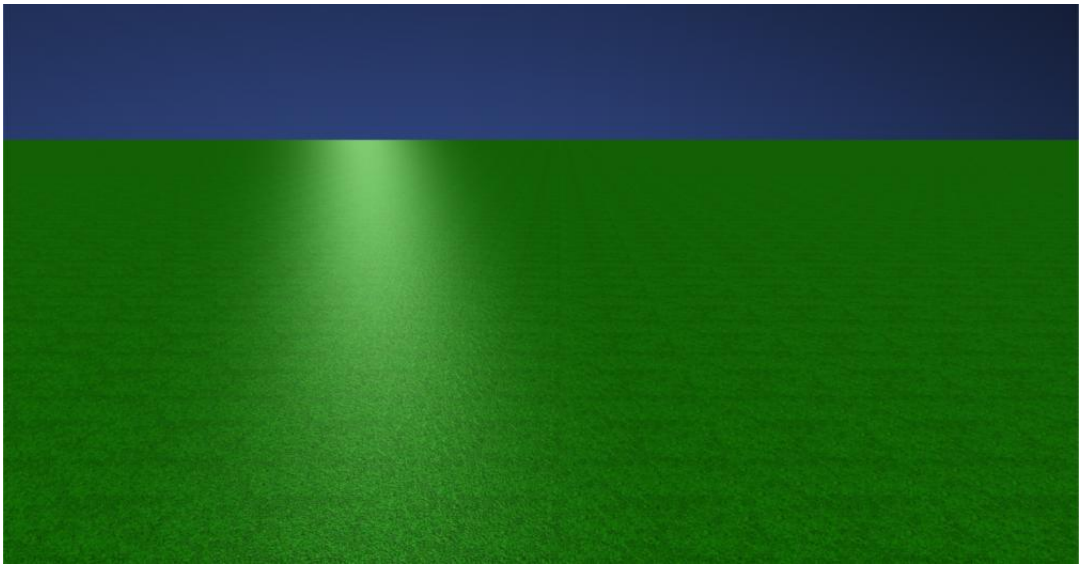

Figure 1: Game world

2.1.2 Hays and obstacles

Hays are represented as rectangular cuboid of constant size 1x1x2 with texture of hay (Figure 2).

Obstacles are represented as rectangular cuboid with various sizes. An example of obstacle of size 1x1x4 is showed on Figure 3.

2.1.3 Vehicle

Vehicle model is a model found on the online marketplace TurboSquid (`www.turbosquid.com`). Model represented as tractor vehicle (Figure 4). For performing an animation

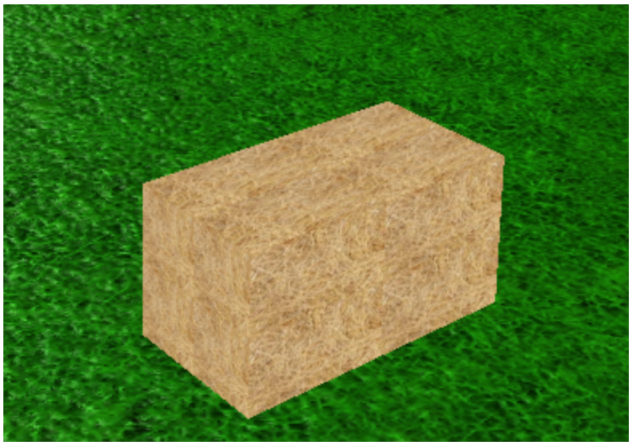

Figure 2: Hay item

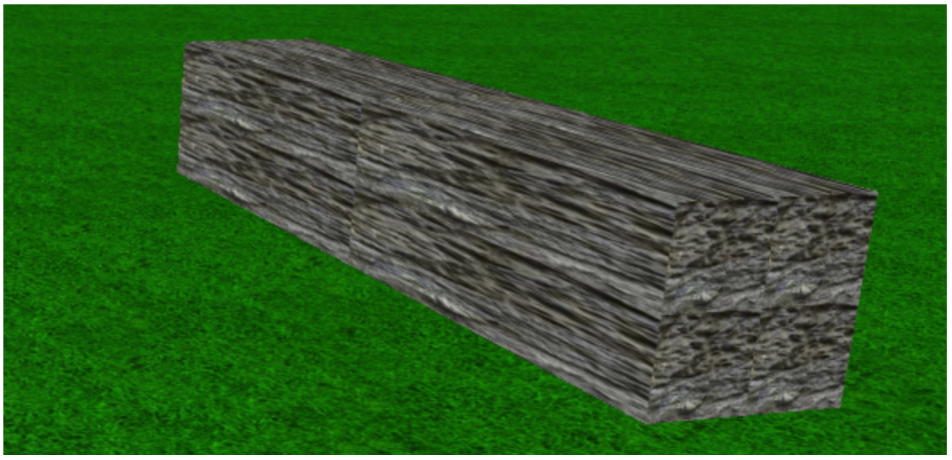

Figure 3: Obstacle item

3D model was separated using Blender software into body, front and rear wheels. Every part of the model is exported through Waveform .obj file format for later using during run time.

2.2 Libraries and Frameworks

All modern browsers has support of WebGL - a JavaScript API for rendering interactive 2D and 3D graphics with GPU acceleration[4]. But usage of WebGL is quite complicated, requiring writing a lot of boilerplate code, helper functions and hardly maintained code. Thus, in this project THREE.js (`https://threejs.org`) open source library was used that offers high-level object-oriented JavaScript API.

Figure 4: Tractor model in Blender

Physics emulation libraries like Physi.js or ammo.js weren't used since they are not necessary for game and make big impact on overall performance.
For debug purposes and performance tracking Stats library (`https://github.com/mrdoob/stats.js`) is used.

2.3 HTML

Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications[3]. Project is represented as a single page web application with *index.html* as main entrance point and consists of two parts:

- Head, which includes data, like name of the web page, author and imports CSS and JavaScript files.
- Body that describes structural hierarchy of UI elements on the web page.

2.4 CSS

CSS (Cascading Style Sheets) is a technology that simplify description and declaration of so called "styles" or parameters of elements of HTML page[2]. "Sino" project contains *styles.css* file under css folder that describes parameters of elements of html page. This includes position inside parent, text color and size, paddings and margins, scales etc.
As a font ArcadeClassic was used.

```
@font-face {
    font-family: 'ArcadeClassic';
    src: url('../font/ARCADECLASSIC.woff');
}
```

Listing 1: Declaration of custom font

2.5 JavaScript

All algorithm parts are written in JavaScript and consist of few separate js files for easy maintenance.

2.5.1 game.js

Entrance point of the game's JavaScript code. Contains game UI initialization, game logic and communication between all other elements of the code. Includes following functions:

- *init* - handles creation and setup all components of the program after start.
- *initScene* - called during init function execution. Initialize and configure main components of THREE.js library renderer and camera.
- *initUI* - called during init function execution. Initialize sreen UI elements like labels, buttons and their callbacks, Stat library.
- *addLight* - alled during init function execution. Initialize, configure and add to scene 3 light source: ambient light using lib THREE.js class *AmbientLight* for creating uniform light all along the scene and 2 directional lights from behind and front using class *DirectionalLight*.

```
    renderer.setSize(window.innerWidth, window.innerHeight);
}

function initUI() {
    scoreGameText = document.getElementById('scoreGame');
    welcomeText = document.getElementById('welcomeText');
    timeProgress = document.getElementById('timeProgress');
    timeLeft = document.getElementById('time');

    soundIcon = document.getElementById('soundToggle');
    soundIcon.addEventListener("click", function () {
        settings.setSoundEnabled(!settings.isSoundEnabled());
        updateSoundIcon();
        if (settings.isSoundEnabled()) {
```

Listing 2: Adding light sources

- *initRendering* - called during init function execution. Describe sequence of operations done for rendering each frame. Call a system function *requestAnimationFrame* to subscribe for next frame update.
- *onWindowResize* - called during init function execution. Called each time size of the browser window is changed. Adjust renderer and camera output parameters according to new window dimensions.

```
                let box = new THREE.BoxHelper(obst, 0xffff00);
                scene.add(box);
            }
        }

        welcomeText.hidden = true;
}
```

Listing 3: onWindowResize function

- *update* - called on every frame update call. Updates vehicle position and configuration, check for vehicle collides with hays and obstacles, update score. Collide check is done using THREE.js *BoundingBox* and it's method *intersectsBox*. In case of intersection of bounding boxes of vehicle and item one is counting as colliding.

2.5.2 core.js

Contains main components of the game such as:

- *Processor* - class which is responsible for generating *Level* objects that describe game level characteristics. Those characteristic are include amount of obstacles and hay blocks, their position and orientation.
- *Vehicle* - class that is responsible for instantiating of 3D model of the vehicle, handling calculating position and orientation of it's components.
- *Sound* - utility class for managing of sound playback, encapsulate instantiating, running and releasing of *Audio* objects by providing handy methods *playBG*, *playHit*, *playCollect* for playing of background, hit and collect audio.
- *Settings* - utility class that encapsulate managing settings and work with persistent storage. Provides methods *isSoundEnabled* and *setSoundEnabled* for saving user's audio policy preference.

2.5.3 items.js

items.js file contains methods for instantiating graphical components of the game such as ground, sky, vehicles, hay, obstacles etc.
All items are instances of *Object3D* class that is the base class for most objects in three.js and provides a set of properties and methods for manipulating objects in 3D space [1].
Mesh class that is successor class of *Object3D* represents an entity that has *Geometry* like *PlaneGeometry*, *CylinderGeometry*, *TextGeometry*, *BoxGeometry* etc. to describe structure of entity and material which provides characteristic of surface, it's appearance and interaction with light. Materials can be parametrized with a single color, list of colors or a texture - image pattern that later mapped onto geometry surface.

Ground

The world's ground is a *Mesh* object with *PlaneGeometry* geometry of size 1000x1000 and *MeshPhongMaterial* with texture of grass (Figure 5).

Sky

Sky object is a sphere (using *SphereGeometry*) *Mesh* object of size 10000 that is surrounding ground plane. Material of this sphere is a texture of blue sky showed on Figure 6. To apply material on internal side of the mesh material's parameter side must be set to *THREE.BackSide*. Full listing of the function *getSky* is showed

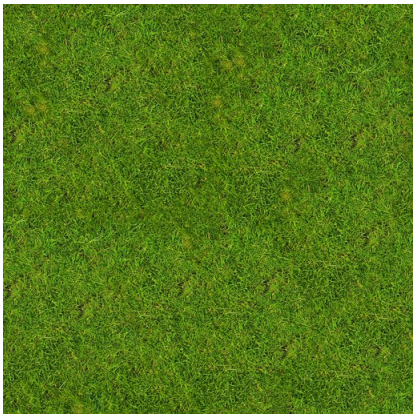

Figure 5: Grass texture

on the Listing 4.

```
function getSky(size) {
    var skyGeo = new THREE.SphereGeometry(size, 25, 25);
    var url = texturesBaseUrl + "sky.jpg";
    var texture = texturesLoader.load(url);
    texture.wrapS = THREE.RepeatWrapping;
    texture.wrapT = THREE.RepeatWrapping;
    texture.repeat.x = size;
    texture.repeat.y = size;
    texture.anisotropy = 16;
    var material = new THREE.MeshPhongMaterial({
        map: texture,
    });
    var sky = new THREE.Mesh(skyGeo, material);
    sky.material.side = THREE.BackSide;
    return sky;
}
```

Listing 4: getSky function

Hay block

Hay blocks are a constant size 1x1x2 cubits *Mesh* with *BoxGeometry* and hay textured material (Figure 7).

Obstacle

Obstacles are a various size cubits *Mesh* with *BoxGeometry* and rock textured material showed on Figure 8.

Figure 6: Sky texture

Figure 7: Hay texture

Figure 8: Sky texture

2.5.4 utils.js

Consist of utility functions that are not depend of the logic and structure of the game.

2.6 Distribution

Final source code of the project is public accessible as git repository on the GitHub web service by `https://github.com/MarcoSchaerfCourses/sino/` link. It could be run online using link `https://marcoschaerfcourses.github.io/sino/` since repository was deployed using GitHub Web Pages service.

3 Known Issues

Testing of final product showed following issues that require additional time to investigation:

- *Performance issues*
  Like any web application and, particularly, those with 3D graphics current project is suffering from performance issues that need more close attention and optimization. Starting from providing loading page experience (screen that showed while models and textures are loading), reducing models complexity and finishing careful memory allocation on every frame update.
- *Texture import*
  There are issues with correct loading textures for vehicle, obstacles and hey blocks that leads to visual glitches and inconvenience. This requires more careful work with professional graphical software and creation more complicated texture files.
- *Bounding Box size*
  Current implementation of *BoundingBox* class doesn't consider 3D object rotation during processing of object's bounding box (Figure **??**). This leads to increasing it's size and incorrect behavior and false positive results during collide check. This problem requires implementing so called *Orientable Bounding Box* or *OOB* that will address this issue.

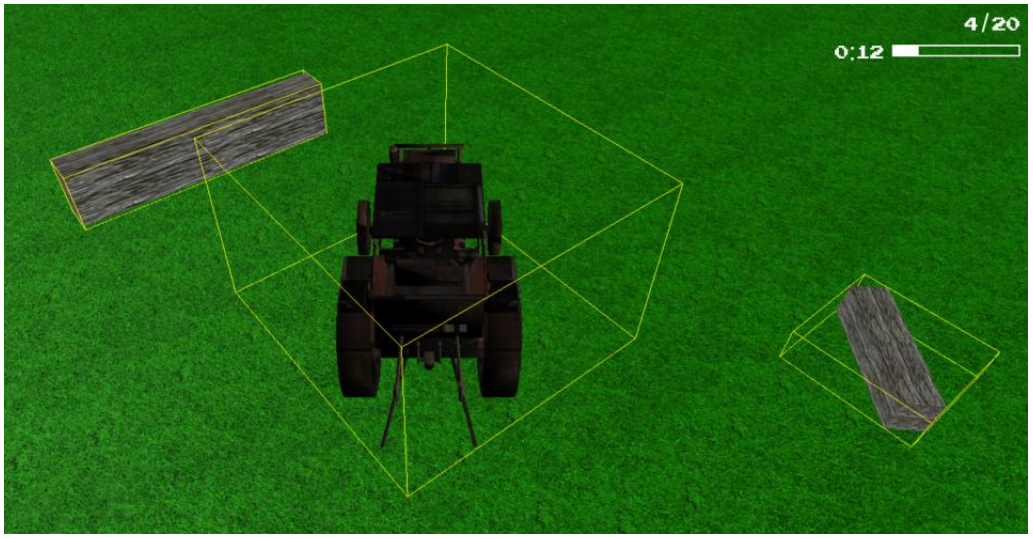

Figure 9: Bounding box size of rotated objects is bigger then their real size

- *Adaptation for small screen sizes*
  The game isn't optimized for mobile displays that requires adjusting of object's scales and camera angles for better user experience.

References

[1] THREEjs.org. Object3D, 2019. [Online; accessed 12-January-2019].

[2] Wikipedia, The Free Encyclopedia. CSS, 2019. [Online; accessed 12-January-2019].

[3] Wikipedia, The Free Encyclopedia. HTML, 2019. [Online; accessed 12-January-2019].

[4] Wikipedia, The Free Encyclopedia. WebGL, 2019. [Online; accessed 12-January-2019].